Sistemas Informáticos
Curso 2004-2005

Seguridad en Globus Toolkit 4

Jorge Báez
Martín de Miguel

Dirigido por:
Prof. Rubén Santiago Montero Dpto
Dpto. DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

Facultad de Informática
Universidad Complutense de Madrid

Índice

1. Introducción

1.1 Objetivos del proyecto

Los objetivos fijados desde el inicio del proyecto son:

- Investigación del funcionamiento del contenedor Globus Toolkit 4.
- Despliegue de servicios web en el contenedor.
- Operación y configuración de seguridad.
- Analizar las posibilidades del sistema actual de seguridad implementado y proponer soluciones para establecer niveles de seguridad de granularidad fina.

1.2 Resumen en español

El proyecto de sistemas informáticos desarrollado consiste en la investigación de la nueva plataforma para servicios distribuidos en grid Globus Toolkit 4. Lo que se persigue es conocer hasta dónde llega su versatilidad, su grado de madurez y si cumple con las expectativas de seguridad necesarias para iniciar una implementación de servicios de computación específicos.

1.3 Resumen en inglés

This project, developed under the subject Computer Systems, is mainly focused on the investigation of the new distributed services grid container Globus Toolkit 4. It is kindly to know how long GT4 gets to, its degree of maturity and if it meets the expectations regarding the security system to start implementing specific grid computing services.

2. La tecnología Grid y Globus Toolkit

2.1 ¿Qué es la tecnología GRID?

En pocas palabras, la tecnología GRID proporciona servicios compartidos de procesamiento de ordenadores y almacenamiento de datos sobre Internet. Esto quiere decir que las tareas que se manden ejecutar sobre tecnología GRID utilizan los recursos que no se aprovechan en otros ordenadores para llevarlas a cabo.

Por tanto al final lo que se tiene es una supercomputadora a nivel mundial que se ocupa de ejecutar las tareas que se le van llegando. Al igual que el servicio WEB en el se encuentran servidores repartidos por todo el mundo ofreciendo información el servicio GRID también está compuesto por una extensa red de ordenadores por todo el mundo desde pc's de particulares, pasando por profesionales más especializados hasta instituciones e universidades que juntos forman una supercomputadora virtual disponible para todos ellos.

La tecnología GRID no sólo trata de compartir los recursos de la CPU sino que también trata la creación de una infraestructura distribuida. Por tanto aparte de englobar el concepto de sistemas operativos distribuidos engloba conceptos como redes de ordenadores

Con el siguiente ejemplo se entenderá de forma práctica cómo puede ser de beneficioso la tecnología GRID.

Un grupo de científicos está estudiando los efectos del gas invernadero sobre el clima a nivel mundial. Los integrantes del grupo se encuentran repartidos por todo el mundo.

La situación sin la tecnología GRID sería la siguiente:

- Mediante correo electrónico se mandan todo tipo de documentación.
- Para mandarse ficheros pesados deben autenticarse en la red donde se encuentren.
- Para poder realizar las pruebas finales del estudio tienen que ejecutar sobre una máquina los

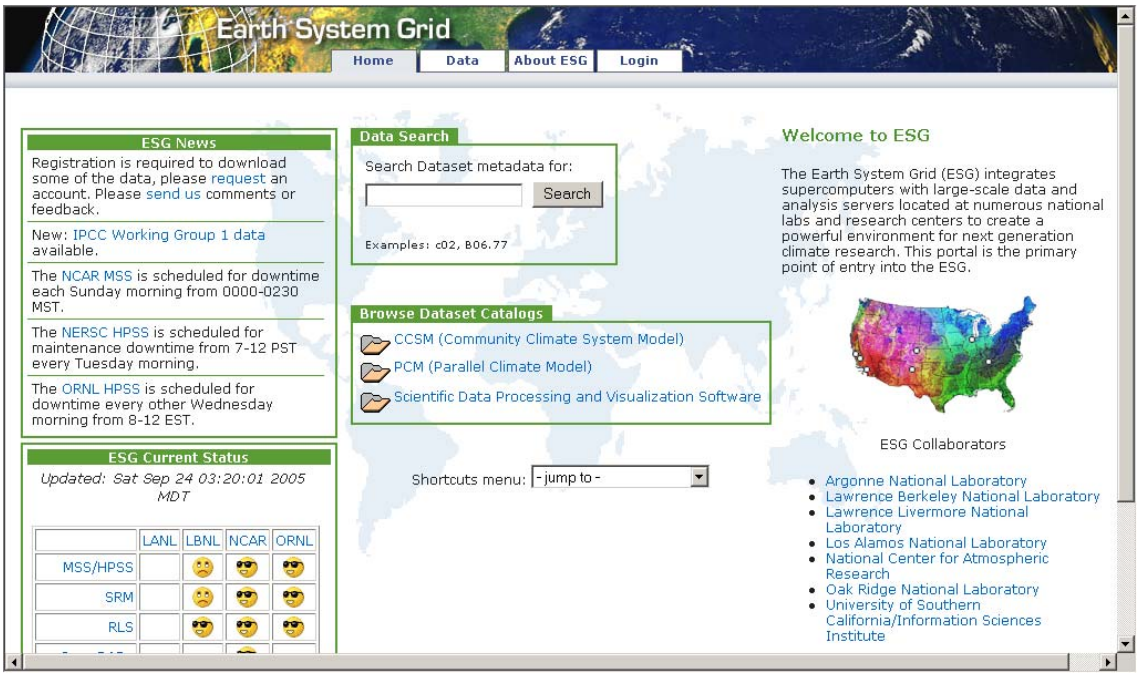
datos que han ido procesando durante la investigación.

- Ahora si introducimos la tecnología GRID los científicos podrían realizar las mismas tareas pero con las siguientes ventajas:
- Comunicación como si estuvieran todos en la misma red.
- Los ficheros pesados de datos ya no tardan tanto como antes al enviarse de un científico a otro al compartirse los recursos necesarios para el envío de datos entre más máquinas.
- Ahora para ejecutar el modelo con los datos del estudio no hace falta que los datos se encuentren todos juntos sino que pueden encontrarse en diferentes máquinas repartidas por todo el mundo.
- Y sobretodo el grupo de científicos podrá acabar mucho antes la ejecución del modelo estudiado ya que podrán ejecutar el modelo más rápidamente obteniendo los recursos libres de procesamiento que se encuentren repartidos por todo el mundo.

Las siguientes aplicaciones son ejemplos reales que utilizan tecnología GRID tanto en el mundo científico como en el mundo de los negocios

- Astronomía :
  - SETI@home: Es un experimento científico que utiliza ordenadores conectados a internet para la Búsqueda de Inteligencia Extraterrestre (siglas en inglés de SETI). Actualmente el experimento utiliza el procesamiento de 3 millones de ordenadores personales repartidos por todo el mundo y hasta ahora ha procesado información equivalente a 600.000 años procesa.
  - NVO: National Virtual Observatory, www.us-vo.org, ofrece datos y recursos de ordenadores mediante tecnología GRID permitiendo a los astrónomos explorar y analizar la inmensidad de datos disponibles en el mundo entero para comprender la formación y evolución del universo.

- Estudios Climáticos :
  - Earth System Grid : https://www.earthsystemgrid.org/ . Portal web que ofrece un entorno de trabajo para las investigaciones climáticas. Está integrado por un conjunto de supercomputadoras, laboratorios con servidores de análisis y centros de investigación que interactúan entre sí para poder aprovechar los recursos del conjunto del entorno formado para así poder realizar estudios más precisos.

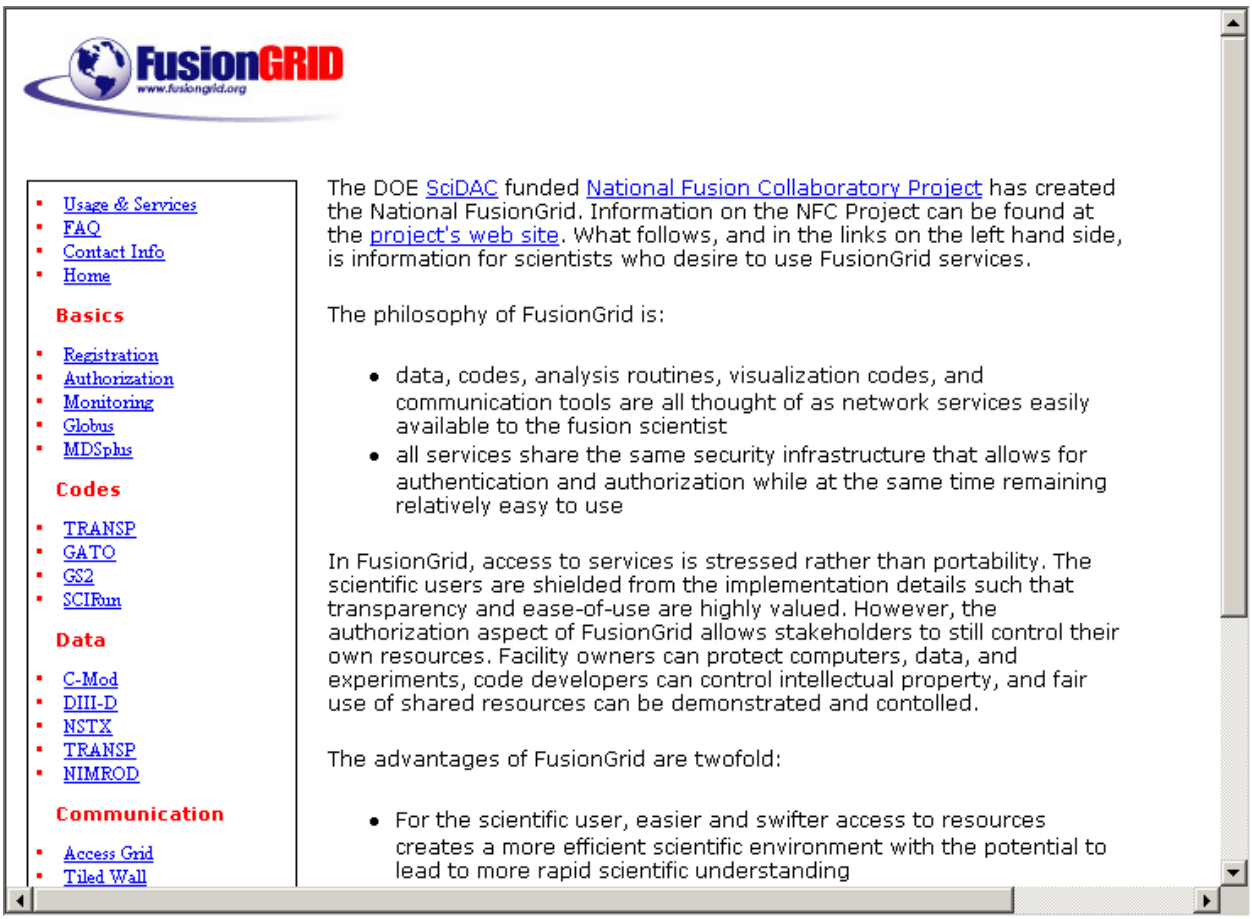


- Colaboración
  - http://www.accessgrid.org/
    Conjunto de recursos como dispositivos multimedia de gran formato, entornos de presentaciones e interfaces para unir servicios GRID para los entornos de visualización. Los recursos son utilizados para soportar las interacciones entre grupos a través del GRID. Por ejemplo, son utilizados mediante el Access GRID para reuniones distribuidas a gran escala, sesiones de trabajo de colaboración, seminarios, tutoriales, etc. Por tanto se diferencian de aquellas herramientas entre escritorios por el tipo de comunicación que no es individual.

- http://www.fusiongrid.org/

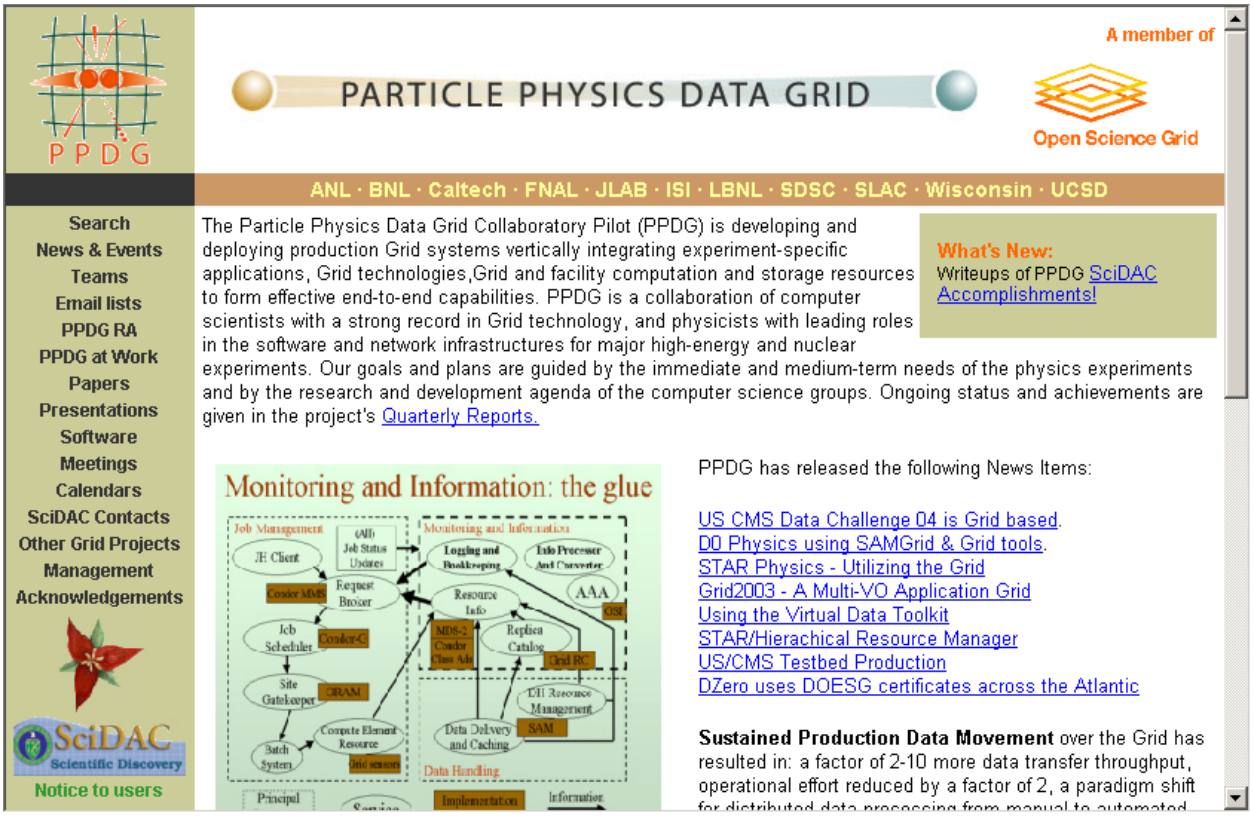
Los principios de FusionGrid son los siguientes :

- Servicios fácilmente accesibles como si de una red propia se tratara para los servicios de datos, códigos, rutinas de análisis, código de visualización.
- Todos los servicios compartidos siguen las mismas políticas de seguridad de autenticación y autorización para permitir un uso homogéneo de los servicios.



- http://www.ppdg.net/

Piloto de colaboración que ofrece todo tipo de servicios GRID. El proyecto de colaboración se crea por la necesidad de suministrar a los experimentos físicos y nucleares y otros de energías del futuro de gran poder energético de modelos de computación distribuidas por todo el mundo para de esta manera obtener el potencial de proceso necesario para los cálculos de enormes dimensiones de los proyectos citados.



2.2 ¿Qué es Globus Alliance?

El Globus Alliance es una comunidad de organizaciones y personas individuales que centraliza y organiza el desarrollo de la tecnología GRID, así como los estándares y los sistemas que lo forman.

Las principales organizaciones que forman la comunidad Globus Alliance son las siguientes:

- Universidad de Chicago (EEUU)
- Laboratorio Nacional de Argonne del Departamento de Energía de EEUU.
- Universidad de Edimburgo (Escocia)
- Centro Sueco de Computadoras en Paralelo
- Centro Nacional de Aplicaciones de Supercomputadoras (EEUU)

Así como la web ha revolucionado el acceso a la información, el Globus Alliance se propone lograr un resultado similar desde el aspecto de la computación. Con el desarrollo de la tecnología GRID desde esta comunidad se propone la posibilidad de nuevos tipos de aplicaciones: acceso a supercomputadoras virtuales, imágenes en vivo desde satélites, almacenamiento masivo de datos y convertir otros recursos online en herramientas sencillas de usar a través de la web.

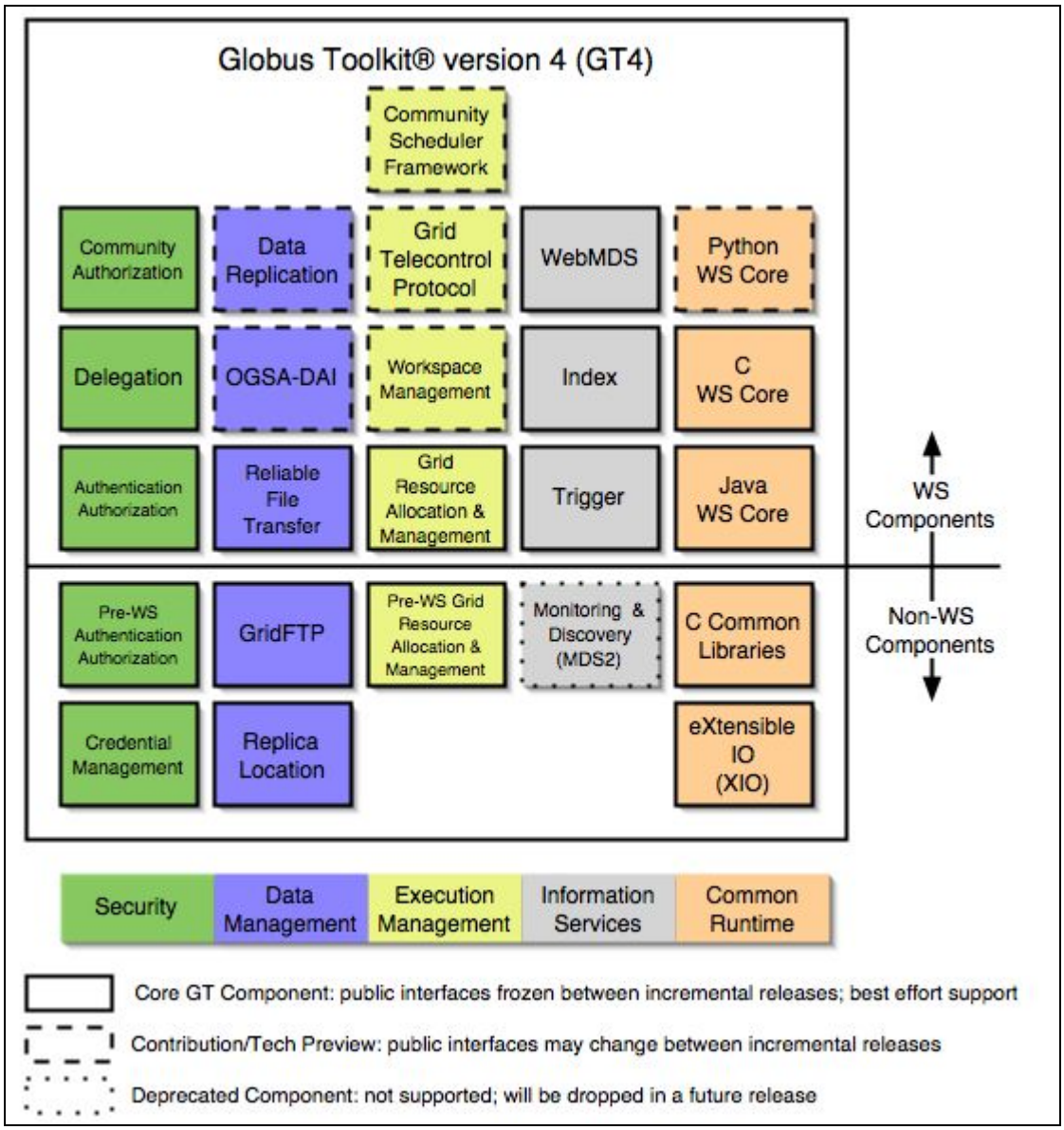
La principal aportación de la Globus Alliance a la comunidad GRID es el proyecto Globus Toolkit.

2.3 ¿Qué es Globus Toolkit?

El Globus Toolkit es un conjunto de herramientas software de código abierto para construir aplicaciones grids. La aportación al desarrollo de la herramienta no sólo es soportada por el Globus Toolkit sino que es apoyada y soportada por más desarrolladores por todo el mundo.

Cada vez son más los proyectos y las compañías que utilizan el Globus Toolkit para aprovechar el potencial que ofrece la red GRID.

El siguiente gráfico muestra los componentes que forman el Globus Toolkit 4 (última versión estable):



Los componentes que forman el Globus Toolkit son los siguientes:

- Seguridad: Autorizaciones, gestión de las identificaciones, delegaciones, etc.
- Gestión de los datos: Servicio FTP, Servicio de replicación,…
- Gestión de ejecución: Localización y gestión de los recursos a ejecutar.
- Servicios de información: Monitorización de los servicios.
- Elementos del core runtime: Distintas librerías con el core de distintos lenguajes.

Los distintos componentes están dentro del Globus Toolkit de manera independiente de forma que se puedan usar por separado o todos juntos a la vez. Al tener cada organización una forma única de operar imposibilita que pueda cooperar con otras organizaciones de forma directa. Globus Toolkit permite eliminar estos obstáculos entre las distintas organizaciones, ofrece servicios, interfaces y protocolos que permiten a los usuarios acceso remoto a recursos tanto estén dentro de su organización como si se encuentran fuera de ésta.

Al tratarse de un conjunto de herramientas de código abierto su crecimiento y expansión ha sido similar al del sistema operativo Linux, sin licencia propietaria cada organización lo adoptaba y lo mejoraba aportando su código al de la comunidad.

Al inicio de la tecnología GRID los científicos e ingenieros de EEUU utilizaban el grid para conectarse a supercomputadoras virtuales y así poder obtener recursos de alto rendimiento para los cálculos necesarios para sus investigaciones. En el año 1996 se inició el proyecto Globus en Argonne, ISI y en la Universidad de Chicago. Actualmente el proyecto renombrado Globus Alliance ha incorporado a otros miembros como la Universidad de Edimburgo, Instituto Real de Tecnología de Suecia, Centro Nacional de Aplicaciones de Supercomputadoras (EEUU) y la corporación Univa. También se unieron como patrocinadores las siguientes instituciones y empresas: NSF, DARPA, NASA, IBM y Micrososft

Dentro del mundo profesional ha surgido bastante interés la tecnología GRID. La idea de convertir la Web en una escritorio de trabajo avanzado a través del Globus Toolkit ha atraído a numerosas empresas: Avaki, DataSynapse, Entropia, Fujitsu, Hewlett-Packard, IBM, NEC, Oracle, Platform, Sun.

Con la aparición del sector profesional dentro del mundo GRID, Globus Toolkit asegura un conjunto de objetivos que preservan la comunidad GRID: garantizar el código abierto de las herramientas existentes y garantizar el ánimo no lucrativo de la iniciativa original del Globus. Por supuesto garantizar estos objetivos obliga a seguir con los anteriores objetivos como es mantener un estándar abierto para favorecer el crecimiento de la tecnología GRID.

3. Modelo de componentes y construcción de servicios WEB con GT4

A continuación se van a exponer los distintos patrones con los que se puede implementar un servicio web con Globus Toolkit4.

3.1 Servicio WEB con estado

3.1.1 Interfaz del servicio

Como primer paso para crear un servicio web stateful se tiene que crear un interfaz del servicio a crear. Es decir, se crea un fichero con los servicios que se van a proporcionar al mundo exterior, estos servicios van a ser las operaciones que proporcionaremos al grid para el resto de usuarios. Este interfaz de servicio se llama en inglés 'portType'.

Para indicar al servicio web qué operaciones se van a ofrecer se utiliza el lenguaje XML : Web Service Description Language (WSDL). Para entender perfectamente la idea expuesta el contenido de este xml se puede comparar con un interfaz de java, se definen los métodos que tienen que implementar las clases que implementan este interfaz.

Por ejemplo, para un servicio con una operación tendríamos el equivalente interfaz java siguiente:

```
public interface Math
{
  public void add(int a);
}
```

Y el descriptor de despliegue del servicio web:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"

targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_inst
ance"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />


<!--============================================================

                     T Y P E S

  ============================================================-->
<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_inst
ance"

xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


      <!-- REQUESTS AND RESPONSES -->

      <xsd:element name="add" type="xsd:int"/>

      <!-- RESOURCE PROPERTIES -->

      <xsd:element name="Value" type="xsd:int"/>
      <xsd:element name="MathResourceProperties">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>
      </xsd:element>

</xsd:schema>
</types>


<!--============================================================
```

```
                    M E S S A G E S

  ============================================================-->
<message name="AddInputMessage">
      <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">

<!--============================================================

                    P O R T T Y P E

  ============================================================-->
<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty"
    wsrp:ResourceProperties="tns:MathResourceProperties">

      <operation name="add">
            <input message="tns:AddInputMessage"/>
            <output message="tns:AddOutputMessage"/>
      </operation>

      </portType>

</definitions>
```

Se utilizan extensiones del pre-procesador del WSDL para no tener que copiar y pegar directamente en el fichero a crear algunos portTypes ya existentes en el fichero oficial WSRF del WSDL.

Los recursos a utilizar se declaran en el apartado <types> :

Un aspecto interesante acerca del WSDL es la no inclusión en absoluto del lenguaje en el cual se ha desarrollado el servicio web o el programa cliente del servicio.

Para enlazar el WSDL con el código del servicio correspondiente el GT4 (Globus Toolkit4) genera automáticamente unas clases stubs que traduce la comunicación necesaria entre el globus y el lenguaje utilizado.

Para que el fichero WSDL pueda conectarse con las clases stub se tiene que generar esta relación en el fichero de mapeos (mappings file) en el cual se relacionan los namespaces con los paquetes Java si es el lenguaje que se está utilizando para desarrollar el servicio.

Nota: Cada mapping tiene que situarse obligatoriamente en una línea.

Por ejemplo:

```
http\://www.globus.org/namespaces/examples/core/MathService_instance=
    org.globus.examples.stubs.MathService_instance
http\://www.globus.org/namespaces/examples/core/MathService_instance/bindings=
    org.globus.examples.stubs.MathService_instance.bindings
http\://www.globus.org/namespaces/examples/core/MathService_instance/service=
    org.globus.examples.stubs.MathService_instance.service
```

El nombre del primer mapeo se tiene que establecer en el fichero WSDL y los otros dos los genera directamente el GT4.

3.1.2 Implementación del servicio

Una vez creado el qué se va hacer mediante el interfaz del servicio el siguiente paso es la implementación del servicio.

Para hacer la vida del programador más fácil a la hora de desarrollar el servicio existe un interfaz de nombres cualificados (qualified name, Qname) para referirse a algo acerca del servicio y de esta manera usar sólo las constantes java para evitar equívocos al usar el namespaces .

Ejemplo QNames:

```
package org.globus.examples.services.core.first.impl;

import javax.xml.namespace.QName;

public interface MathQNames {
	public static final String NS =
"http://www.globus.org/namespaces/examples/core/MathService_instance";

	public static final QName RP_VALUE = new QName(NS, "Value");

	public static final QName RESOURCE_PROPERTIES = new QName(NS,
			"MathResourceProperties");
}
```

La implementación del servicio usando el patrón Stateful crea en una sola clase java el servicio y el rescurso. Este tipo de uso implementado el patrón Stateful sólo se recomienda utilizarlo para desarrollos de servicios básicos. Es recomendable si el servicio es más complejo separar el recurso y el servicio en dos clases separadas como se implementa en el patrón Singleton.

Ejemplo Servicio:

```
package org.globus.examples.services.core.first.impl;

import java.rmi.RemoteException;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;
import org.globus.examples.stubs.MathService_instance.AddResponse;
import org.globus.examples.stubs.MathService_instance.SubtractResponse;
import org.globus.examples.stubs.MathService_instance.GetValueRP;

public class MathService implements  Resource, ResourceProperties {

	/* Resource Property set */
	private ResourcePropertySet propSet;

	/* Resource properties */
	private int value;

	/* Constructor. Initializes RPs */
	public MathService() throws RemoteException {
		/* Create RP set */
		this.propSet = new SimpleResourcePropertySet(
				MathQNames.RESOURCE_PROPERTIES);
```

```
            /* Initialize the RP's */
            try {
                  ResourceProperty valueRP = new ReflectionResourceProperty(
                              MathQNames.RP_VALUE, "Value", this);
                  this.propSet.add(valueRP);
                  setValue(0);

            } catch (Exception e) {
                  throw new RuntimeException(e.getMessage());
            }
      }

      /* Get/Setters for the RPs */
      public int getValue() {
            return value;
      }

      public void setValue(int value) {
            this.value = value;
      }

      /* Remotely-accessible operations */

      public AddResponse add(int a) throws RemoteException {
            value += a;

            return new AddResponse();
      }

      public SubtractResponse subtract(int a) throws RemoteException {
            value -= a;

            return new SubtractResponse();
      }

      public int getValueRP(GetValueRP params) throws RemoteException {
            return value;
      }

      /* Required by interface ResourceProperties */
      public ResourcePropertySet getResourcePropertySet() {
            return this.propSet;
      }
}
```

- Lo primero a describir según el patrón Stateful es que la clase del servicio tiene que implementar el recurso y el servicio por lo que se implementan Resource, ResourceProperties.

```
public class MathService implements  Resource, ResourceProperties
```

Entonces según el interfaz ResourceProperties se tiene que implementar el siguiente método obligatoriamente:

```
private ResourcePropertySet propSet;

public ResourcePropertySet getResourcePropertySet() {
      return this.propSet;
}
```

- Al tener una propiedad el recurso a implementar (value) se crea un atributo en la clase servicio de java con sus correspondientes get/set para que el servicio pueda obtener/establecer automáticamente su valor:

```
/* Resource properties */
private int value;

/* Get/Setters for the RPs */
public int getValue() {
	return value;
}

public void setValue(int value) {
	this.value = value;
}
```

Se corresponde con el fichero WSDL con las siguientes líneas:

```
<xsd:element name="Value" type="xsd:int"/>
```

- El siguiente paso es implementar el constructor, en el cual se inicia los recursos de los QNames y se inicializan los valores del recurso Value

```
/* Constructor. Initializes RPs */
public MathService() throws RemoteException {

	this.propSet = new SimpleResourcePropertySet(
			MathQNames.RESOURCE_PROPERTIES);

	/* Initialize the RP's */
	try {

		ResourceProperty valueRP = new ReflectionResourceProperty(
				MathQNames.RP_VALUE, "Value", this);
		this.propSet.add(valueRP);
		setValue(0);

	} catch (Exception e) {
		throw new RuntimeException(e.getMessage());
	}
}
```

- Y finalmente se implementa el método que será llamado de forma remota ante cualquier petición del servicio:

```
public AddResponse add(int a) throws RemoteException {
	value += a;
	return new AddResponse();
}
```

Y el método que devuelve el valor de forma remota:

```
public int getValueRP(GetValueRP params) throws RemoteException {
	return value;
}
```

3.1.3 Despliegue del servicio web

Hasta ahora se ha creado un interfaz del servicio (WSDL) y la implementación del servicio web. Ahora se tiene que configurar el contenedor de servicios web para instalar nuestro nuevo servicio y que esté disponible para ser llamado.

- Descriptor de despliegue (deployment descriptor) fichero.wsdd

El siguiente código de fichero contiene la configuración de los servicios web en el contenedor

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="examples/core/first/MathService" provider="Handler"
use="literal" style="document">
        <parameter name="className"
value="org.globus.examples.services.core.first.impl.MathService"/>

<wsdlFile>share/schema/examples/MathService_instance/Math_service.wsdl</wsdlFile
>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

El tag `<service name>` indica el nombre del servicio y la URI entera obteniendo el contexto base del contenedor de los servicios sería algo parecido a:

```
http://localhost:8080/wsrf/services/examples/core/first/MathService
```

El tag `ClassName` indica la ruta con paquete incluida de la clase que implementa el servicio.

El tag `<wsdlFile>` indica dónde tiene que recoger el contenedor el fichero con la información del interfaz del servicio.

Si se quiere que el servicio se levante al arrancar el contenedor se establece el valor `true` en el tag `<loadOnStartup>`.

3.1.4 Creación del GAR

Todos los ficheros generados hasta ahora se organizan en una librería donde el contenedor de los servicios web puede obtenerlos de manera más rápida y directa. Esta librería se llama fichero gar (Grid Archive) y será generado de forma automática por el GT4 mediante el uso de la herramienta ANT.

En la generación del fichero gar se realizan las siguientes tareas:

- Proceso del fichero WSDL.
- Creación de las clases stubs desde el WSDL.

- Compilación de las implementaciones de los servicios.
- Organización de los ficheros en una estructura específica.

3.1.5 Despliegue del servicio web

Para realizar el despliegue del servicio web en el contenedor el usuario a realizar esta operación tiene que tener permisos de escritura en el directorio de Globus ya que desempaqueta y copia los ficheros en directorios adecuados.

Despliegue:

```
globus-deploy-gar $EXAMPLES_DIR/org_globus_examples_services_core_first.gar
```

Eliminar servicio:

```
globus-undeploy-gar org_globus_examples_services_core_first
```

Para realizar estas operaciones el contenedor de servicios web tiene que estar parado para que al arrancar de nuevo tenga en cuenta los cambios realizados.

3.1.6 Creación de un cliente

Para comprobar que nuestro servicio se ha desarrollado correctamente crearemos un cliente de prueba que realice una petición de nuestro servicio implementado.

```
package org.globus.examples.clients.MathService_instance;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import org.globus.examples.stubs.MathService_instance.MathPortType;
import org.globus.examples.stubs.MathService_instance.GetValueRP;
import
org.globus.examples.stubs.MathService_instance.service.MathServiceAddressingLoca
tor;

public class Client {

       public static void main(String[] args) {
              MathServiceAddressingLocator locator = new
MathServiceAddressingLocator();

              try {
                     String serviceURI=args[0];

                     EndpointReferenceType endpoint = new
EndpointReferenceType();
                     endpoint.setAddress(new Address(serviceURI));

                     MathPortType math = locator.getMathPortTypePort(endpoint);

                     // Perform an addition
                     math.add(10);

                     // Perform another addition
                     math.add(5);

                     // Access value
                     System.out.println("Current value:" + math.getValue(new
```

```
GetValueRP()));

                    // Perform a subtraction
                    math.subtract(5);

                    // Access value
                    System.out.println("Current value:" + math.getValue(new
GetValueRP()));
             } catch (Exception e) {
                    e.printStackTrace();
             }
       }
}
```

Se crea una referencia del servicio así mismo (ya que tiene el recurso él mismo) con la clase `EndpointReferenceType`

A continuación se obtiene una referencia del interfaz del servicio (portType) a través de la clase `MathPortType` mediante la instanciación de una locator de la referencia anteriormente creada del recurso `(EndpointReferenceType)`

Una vez que se tiene referencia al servicio se ejecuta la operación que dispone el servicio: `math.add(10);`

Se puede comprobar el estado del recurso mediante la operación `math.getValue(new GetValueRP())`

Una vez que se ha compilado el cliente y todos los pasos anteriores se han completado de forma correcta se puede arrancar el contenedor de servicios web mediante la siguiente instrucción sin seguridad: `globus-start-container –nosec.` Al arrancar el contenedor se puede observar los servicios que se encuentran disponibles.

3.2 Modelo Singleton

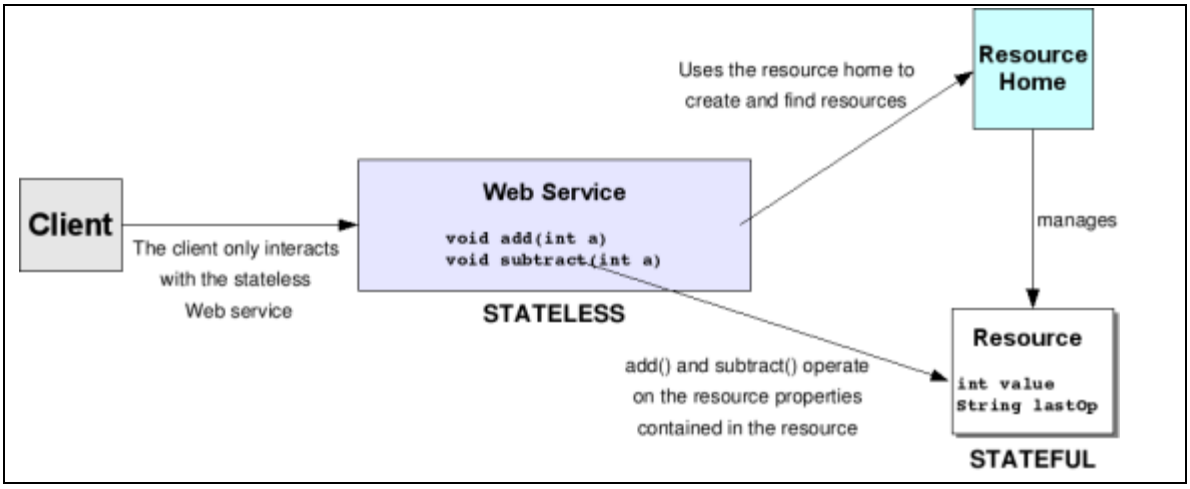
Es el patrón para implementar servicios web más utilizados dentro del Globus Toolkit. Resumidamente se sigue teniendo un único recurso como el servicio stateful pero esta vez se separa el servicio y el recurso en dos clases.

3.2.1 Separación de la implementación

Se crean nuevas entidades para separar el recurso de la implementación:

- Recurso: La entidad que representa el recurso en sí mismo.
- Home: Entidad para crear y encontrar accesos.
- Servicio: Implementación del servicio.

El siguiente gráfico muestra la relación entre sí de las distintas entidades:



El cliente siempre accede al recurso con datos a través del servicio web stateless, el cual para localizar el recurso accede a la Home.

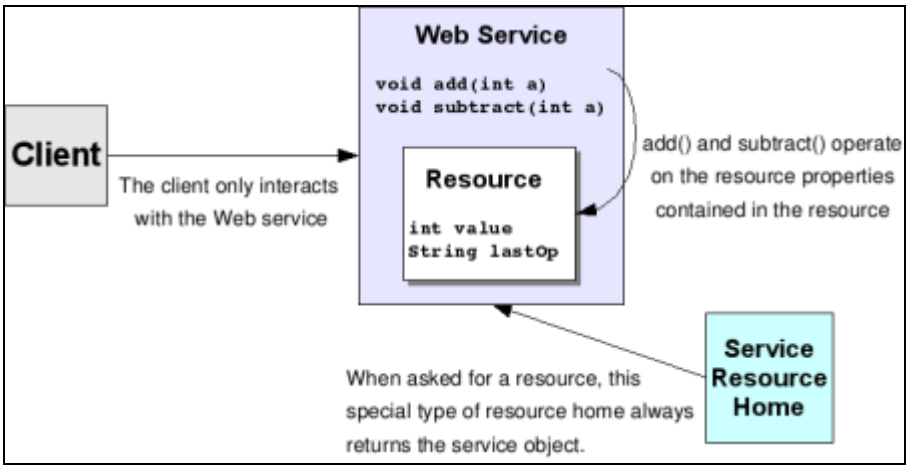
El servicio a diferencia de la implementación stateful es stateless: el cliente sólo interactúa con la clase implementación del servicio.

El recurso es una clase stateful con la información que se quiere mantener.

El home es la entidad encargada de buscar y recuperar los recursos.

ServiceResourceHome: La clase utilizada para los ejemplos ServiceResourceHome no es la más apropiada para desarrollar servicios más complejos, es un recurso especial proporcionado por Globus para devolver siempre el objeto servicio que preguntó por un recurso, así en el caso de patrón stateful se utiliza este clase al encontrarse en la misma clase el recurso y el servicio

```
<resource name="home" type="org.globus.wsrf.impl.ServiceResourceHome">
```



3.2.2 *Interfaz del servicio (WSDL)*

El fichero WSDL es el mismo que en el caso del patrón stateful

3.2.3 *Interfaz del Namespaces*

Es recomendable cambiar el nombre del paquete con respecto al del patrón stateful, pero el

contenido de las constantes sigue siendo el mismo.

```
package org.globus.examples.services.core.singleton.impl;

import javax.xml.namespace.QName;

public interface MathQNames {
	public static final String NS =
"http://www.globus.org/namespaces/examples/core/MathService_instance";

	public static final QName RP_VALUE = new QName(NS, "Value");

	public static final QName RESOURCE_PROPERTIES = new QName(NS,
			"MathResourceProperties");
}
```

3.2.4 *Recurso*

La clase que se muestra a continuación representa una entidad recurso que a diferencia del ejemplo del patrón stateful no contiene las operaciones del servicio web que se implementan a hora en la clase propia de la implementación del servicio.

```
package org.globus.examples.services.core.singleton.impl;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.SimpleResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;

public class MathResource implements Resource, ResourceProperties {

	/* Resource Property set */
	private ResourcePropertySet propSet;

	/* Resource properties */
	private int value;

	private String lastOp;

	/* Initializes RPs */
	public void initialize() throws Exception {
		this.propSet = new SimpleResourcePropertySet(
				MathQNames.RESOURCE_PROPERTIES);

		try {
			ResourceProperty valueRP = new ReflectionResourceProperty(
					MathQNames.RP_VALUE, "Value", this);
			this.propSet.add(valueRP);
			setValue(0);

		} catch (Exception e) {
			throw new RuntimeException(e.getMessage());
		}
	}

	/* Get/Setters for the RPs */
	public int getValue() {
		return value;
```

```
	}

	public void setValue(int value) {
		this.value = value;
	}

	/* Required by interface ResourceProperties */
	public ResourcePropertySet getResourcePropertySet() {
		return this.propSet;
	}
}
```

3.2.5 *Implementación del servicio*

En esta clase se implementarán los métodos necesarios para crear las operaciones del servicio web. En este caso al tratarse del patrón stateless no podrá acceder de forma directa al recurso sino que tendrá que hacerlo a través de una entidad que represente el recurso, en este ejemplo MathResponse:

```
public AddResponse add(int a) throws RemoteException {
	MathResource mathResource = getResource();
	mathResource.setValue(mathResource.getValue() + a);

	return new AddResponse();
}
```

Para obtener el recurso se implementa un método que devuelve a través del método `ResourceContext.getResourceContext().getResource()` la instancia del recurso.

La clase ResourceContext está desarrollada dentro del Globus:

```
private MathResource getResource() throws RemoteException {
	Object resource = null;
	try {
		resource = ResourceContext.getResourceContext().getResource();
	} catch (Exception e) {
		throw new RemoteException("", e);
	}

	MathResource mathResource = (MathResource) resource;
	return mathResource;
}
```

Al final la clase que implementa el servicio web sería la siguiente:

```
package org.globus.examples.services.core.singleton.impl;

import java.rmi.RemoteException;

import org.globus.examples.services.core.singleton.impl.MathResource;
import org.globus.wsrf.ResourceContext;
import org.globus.examples.stubs.MathService_instance.AddResponse;
import org.globus.examples.stubs.MathService_instance.SubtractResponse;
import org.globus.examples.stubs.MathService_instance.GetValueRP;

public class MathService {
```

```
	/*
	 * Private method that gets a reference to the resource specified in the
	 * endpoint reference.
	 */
	private MathResource getResource() throws RemoteException {
		Object resource = null;
		try {
			resource =
ResourceContext.getResourceContext().getResource();
		} catch (Exception e) {
			throw new RemoteException("Unable to access resource.", e);
		}

		MathResource mathResource = (MathResource) resource;
		return mathResource;
	}

	/* Implementation of add, subtract, and getValue operations */

	public AddResponse add(int a) throws RemoteException {
		MathResource mathResource = getResource();
		mathResource.setValue(mathResource.getValue() + a);

		return new AddResponse();
	}

	public int getValueRP(GetValueRP params) throws RemoteException {
		MathResource mathResource = getResource();

		return mathResource.getValue();
	}

}
```

3.2.6 *El recurso Home*

Implementar la clase Home mediante el GT4 es sencillo de realizar, ya que solamente se tiene que extender la clase que se implemente de la clase Globus SingletonResourceHome.

Ejemplo:

```
package org.globus.examples.services.core.singleton.impl;

import org.globus.wsrf.Resource;
import org.globus.wsrf.impl.SingletonResourceHome;

public class MathResourceHome extends SingletonResourceHome {

	public Resource findSingleton() {
		try {
			// Create a resource and initialize it.
			MathResource mathResource = new MathResource();
			mathResource.initialize();
			return mathResource;
		} catch (Exception e) {
			e.printStackTrace();
			return null;
		}

	}
```

```
}
```

La clase SingletonResourceHome implementa casi toda la funcionalidad necesaria para gestionar recursos únicos. La única cosa que se tiene que realizar es implementar el método findSingleton, que es llamado internamente por la clase anteriormente mencionada ResourceContext. El método findSingleton lo único que realiza es crear la clase de recursos, inicializarla y devolverla. Internamente sólo existe la instancia única de un mismo recurso, la clase SingletonResourceHome se encarga de gestionarla internamente para que no se creen varias instancias del mismo recurso siguiendo el patrón de clases Singleton.

Puede no ser muy útil la utilización de una clase Home para acceder a un único recurso pero en el caso que existan múltiples recursos es muy beneficiosa su utilización ya que gestiona el ciclo de vida de cada recurso.

3.2.7 Ejecución del servicio singleton

Al haber modificado el paquete donde se encuentra el servicio web se tendrá que modificar el nombre del servicio y el nombre de la clase del servicio:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="examples/core/singleton/MathService" provider="Handler"
use="literal" style="document">
        <parameter name="className"
value="org.globus.examples.services.core.singleton.impl.MathService"/>

<wsdlFile>share/schema/examples/MathService_instance/Math_service.wsdl</wsdlFile
>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>

</deployment>
```

En este tipo de patrón tiene más sentido que en el stateful la utilización del fichero de configuración jndi, ya que indentifica dónde se encuentra el Home:

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

<service name="examples/core/singleton/MathService">
      <resource name="home"
type="org.globus.examples.services.core.singleton.impl.MathResourceHome">
      <resourceParams>

            <parameter>
                  <name>factory</name>
                  <value>org.globus.wsrf.jndi.BeanFactory</value>
            </parameter>
```

```
        </resourceParams>

        </resource>
</service>

</jndiConfig>
```

El fichero jndi contiene diferentes servicios cuyo atributo name debe coincidir con el servicio establecido en el fichero de despliegue .wsdd. Este atributo es pues la conexión entre el fichero wsdd y el fichero jndi.

Dentro del tag service se encuentra la definición del recurso home, en el cual se establece la clase que lo representa. En este ejemplo se muestra sólo un recurso home pero lo normal es que haya más de un recurso home.

Comparando el fichero jndi del patrón singleton con el del patrón stateful la única diferencia es la utilización de la clase ServiceResourceHome que proprociona Globus en el patrón stateful

Para probar el servicio creado se tiene que vovler a crear el fichero gar como en el caso del patrón stateful, parar y levantar de nuevo el contenedor para que tenga en cuenta los cambios realizados.

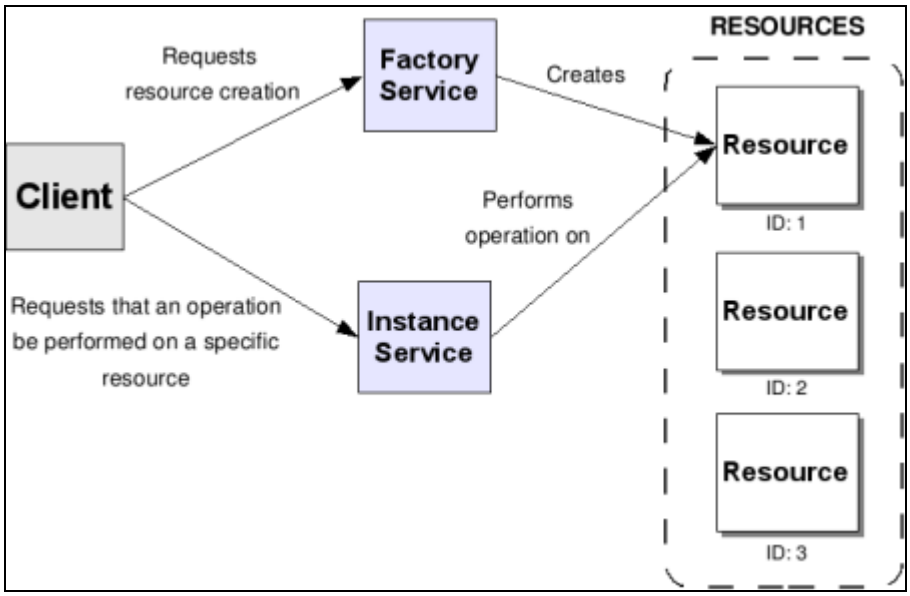
3.3 Patrón Múltiples Recursos

Con el uso de este patrón será posible la utilización de múltiples recursos por parte de un servicio web, para ello se implementará el patrón instancia/factoría.

El patrón instancia/factoría es un diseño bastante extendido dentro de los diseños software, especialmente en diseños orientados a objetos. La idea de este patrón es la siguiente: no se pueden crear directamente instancias de un objeto, la manera de crear instancias de una clase es a través de una factoría que proporciona una operación create

El uso de este patrón está recomendado por el Web Services Resource Framework (WSRF) cuando se tienen que obtener más de un recurso.

El siguiente gráfico muestra la relación entre el cliente, el servicio y los recursos a los que accede:

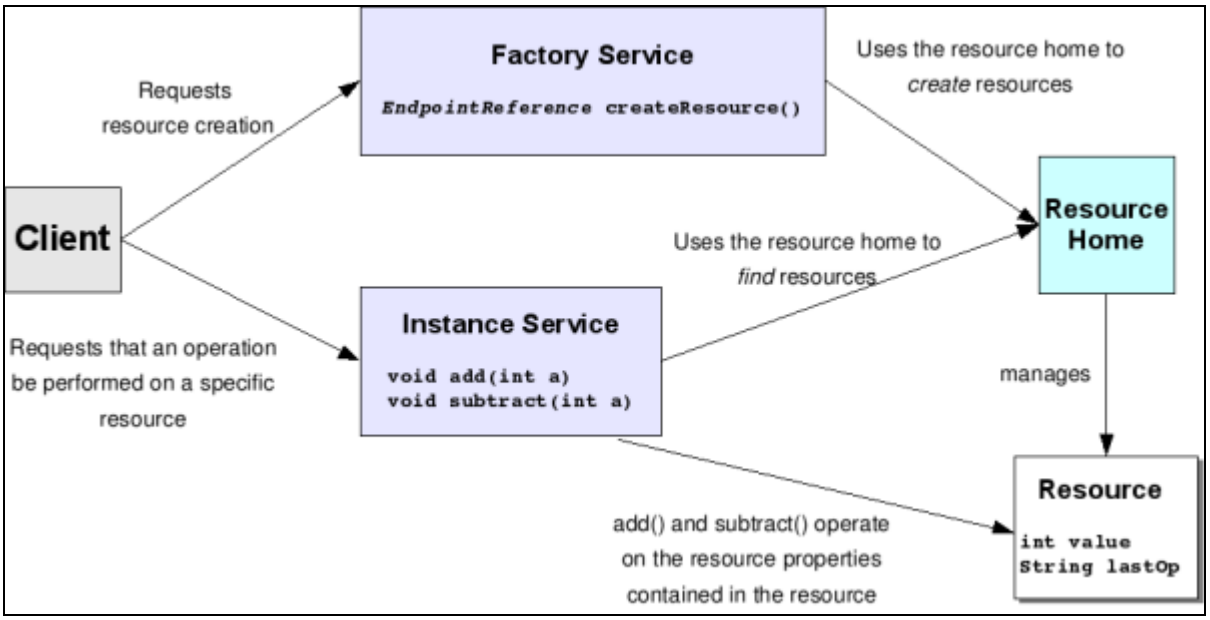


El cliente para acceder a una operación del servicio tiene primero que realizar una petición de creación de una instancia a la factoría, no puede directamente crear una instancia del servicio. La factoría se encarga de crear e inicializar los recursos solicitados para que las instancias de las factorías puedan acceder a ellos.

3.3.1 Implementación del patrón factoría

La implementación del patrón factoría es de igual manera que el patrón singleton pero con dos diferencias:

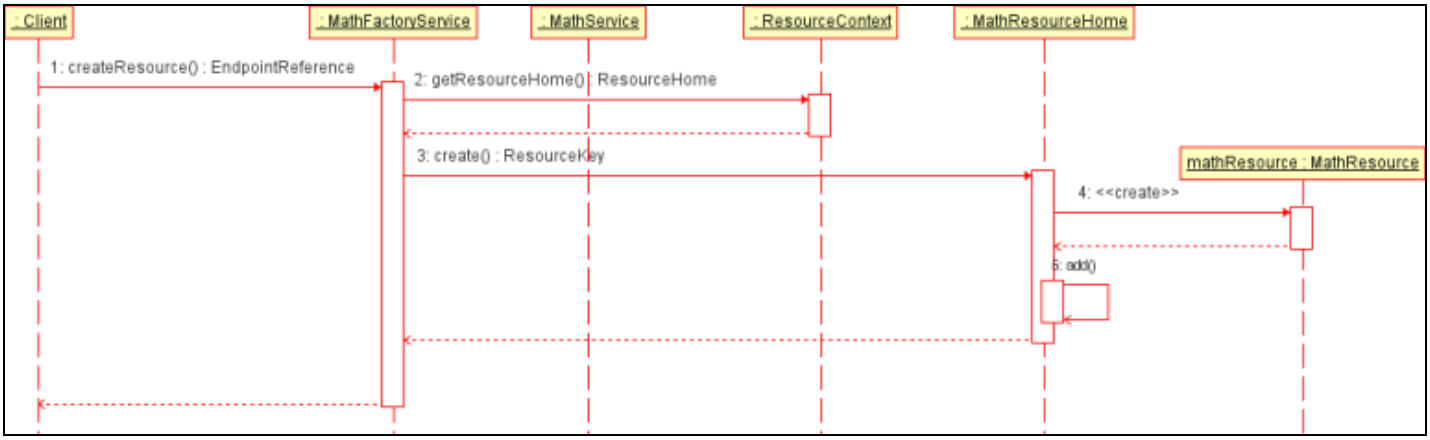
- Servicio factoría y servicio instancia: Para poder gestionar múltiples recursos se necesitarán dos servicios, el servicio factoría y el servicio instancia. La factoría proporciona una operación createResource que devuelve una referencia al recurso creado, EndPointReference (EPR). La instancia proporciona las operaciones del servicio web própiamente dicho, ejemplo:add.
- Home de los recursos: La home de los recursos ya no trata con un único recurso sino que mantiene la relación con múltiples recursos. El servicio de factoría usará la home de recursos para crear nuevos recursos mientras que el servicio de instancia usará la home de recursos para encontrar un recurso por una clave dada.



Entonces para la implementación del patrón factoría se tendrán las siguientes clases:

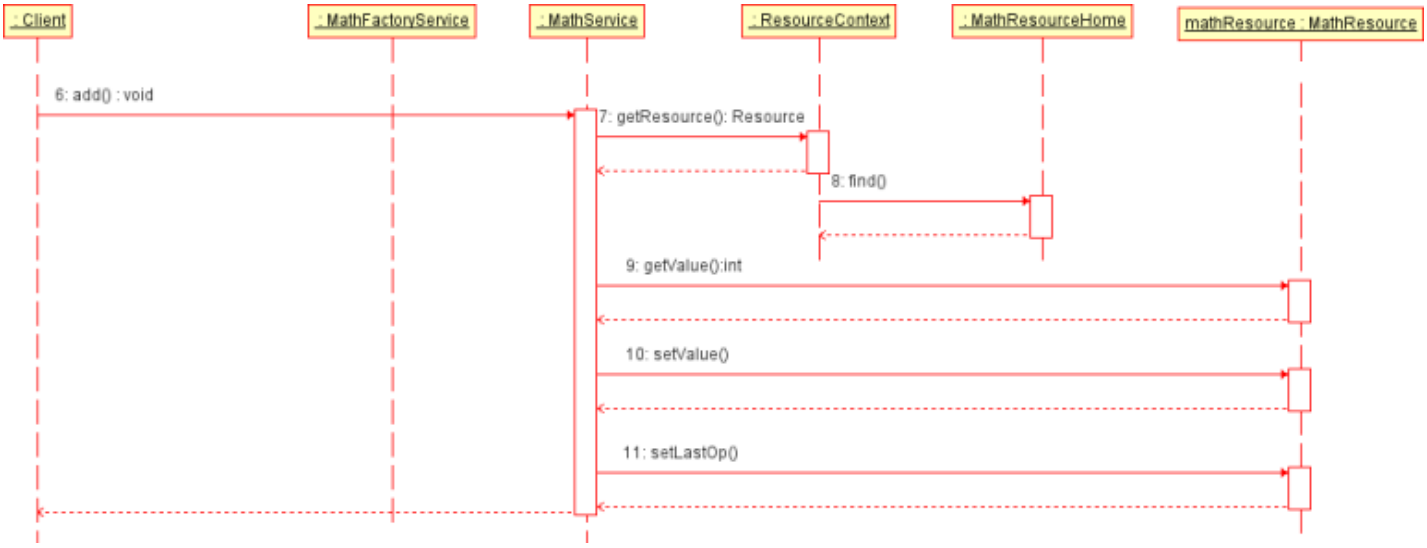
- Servicio Factoría.
- Servicio Instancia.
- Recurso.
- Home del recurso.

Con el siguiente diagrama de secuencias de clases se muestra un flujo de una petición de un cliente para crear un recurso:



Mediante la devolución del objeto ResourceKey al crear un recurso la factoría se accede al recurso y se abilita que pueda realizar operaciones sobre él.

El siguiente diagrama de secuencias de clases muestra la invocación de un recurso:



La clase ResourceContext devuelve el recurso para que después de encontrarlo se pueda realizar sobre él las operaciones establecidas por el cliente.

3.3.2 *El servicio factoría*

La creación del fichero WSDL es muy sencilla, es casi igual que los anteriores patrones, sólo se tiene que añadir una operación createResource sin parámetros que devuelve una referencia endPoint.

Ejemplo de fichero wsdl:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="FactoryService"

targetNamespace="http://www.globus.org/namespaces/examples/core/FactoryService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.globus.org/namespaces/examples/core/FactoryService"
```

```
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


<!--============================================================

                     T Y P E S

  ============================================================-->
<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/examples/core/FactoryService"
    xmlns:tns="http://www.globus.org/namespaces/examples/core/FactoryService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

       <xsd:import
           namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
           schemaLocation="../../ws/addressing/WS-Addressing.xsd" />


       <!-- REQUESTS AND RESPONSES -->

       <xsd:element name="createResource">
              <xsd:complexType/>
       </xsd:element>
       <xsd:element name="createResourceResponse">
       <xsd:complexType>
              <xsd:sequence>
                     <xsd:element ref="wsa:EndpointReference"/>
              </xsd:sequence>
       </xsd:complexType>
       </xsd:element>

</xsd:schema>
</types>


<!--============================================================

                     M E S S A G E S

  ============================================================-->
<message name="CreateResourceRequest">
       <part name="request" element="tns:createResource"/>
</message>
<message name="CreateResourceResponse">
       <part name="response" element="tns:createResourceResponse"/>
</message>



<!--============================================================

                     P O R T T Y P E

  ============================================================-->
<portType name="FactoryPortType">

       <operation name="createResource">
```

```
            <input message="tns:CreateResourceRequest"/>
            <output message="tns:CreateResourceResponse"/>
      </operation>

</portType>

</definitions>
```

Cambios con respecto a los anteriores patrones:

Se actualiza el nombre del nuevo namespaces.

Se tiene que declarar la configuración del namespaces

Se tiene que importar el fichero de esquema con las direcciones del WS.

Se crea la nueva operación para que la factoría pueda crear los recursos

Relación de los namespaces WSDL con los paquetes java (Mapping de los namespaces):

```
http\://www.globus.org/namespaces/examples/core/FactoryService=
                                    org.globus.examples.stubs.Factory
http\://www.globus.org/namespaces/examples/core/FactoryService/bindings=
                                    org.globus.examples.stubs.Factory.bindings
http\://www.globus.org/namespaces/examples/core/FactoryService/service=
                                    org.globus.examples.stubs.Factory.service
```

Nota: Recordadque sólo hay tres líneas, cada mapping en una línea

Implementación java del servicio factoría:

```
public class MathFactoryService {

      /* Implementation of createResource Operation */
      public CreateResourceResponse createResource(CreateResource request)
                  throws RemoteException {

      }
}
```

Obtención del recurso home (dentro del createResource) y uso del recurso home para crear un nuevo recurso:

```
ResourceContext ctx = null;
MathResourceHome home = null;
ResourceKey key = null;
try {
      ctx = ResourceContext.getResourceContext();
      home = (MathResourceHome) ctx.getResourceHome();
      key = home.create();
} catch (Exception e) {
      throw new RemoteException("", e);
}
```

La variable key contiene el identificador del recurso, el cual se utiliza para construir la referencia endpoint:

```
EndpointReferenceType epr = null;

try {
```

```
        URL baseURL = ServiceHost.getBaseURL();
        String instanceService = (String) MessageContext
                        .getCurrentContext().getService().getOption("instance");
        String instanceURI = baseURL.toString() + instanceService;
        // The endpoint reference includes the instance's URI and the resource
key
        epr = AddressingUtils.createEndpointReference(instanceURI, key);
} catch (Exception e) {
        throw new RemoteException("", e);
}
```

Una vez que ya se tiene el endpoint se establece en la respuesta:

```
CreateResourceResponse response = new CreateResourceResponse();
response.setEndpointReference(epr);
return response;
```

3.3.3 *Servicio Instancia*

Prácticamente no cambia la implementación con respecto al anterior patrón de diseño.

Ejemplo: Ahora se devuelve una clase de tipo AddResponse:

```
public AddResponse add(int a) throws RemoteException {
        MathResource mathResource = getResource();
        mathResource.setValue(mathResource.getValue() + a);

        return new AddResponse();
}
```

El método `getResource()` no cambia con respecto al anterior patrón.

La ventaja de utilizar referencias endpoints en vez de utilizar uri's es que se pasan de manera transparente las claves de los recursos, así se pueden declarar métodos como add(int a) en vez de add(int a, int resourceID)

3.3.4 *Recurso*

Sólo se necesitan pequeños cambios para adaptar este patrón con respecto al anterior. Cada recurso tiene una clave única que lo identifica por lo cual se tiene que implementar un interfaz para poder llevar esta funcionalidad a cabo:

```
public class MathResource
                implements Resource, ResourceIdentifier, ResourceProperties
```

El interfaz implementado obliga el desarrollo del método getID para retornar el identificador del recurso:

```
/* Resource key. This uniquely identifies this resource. */
private Object key;

/* Required by interface ResourceIdentifier */
public Object getID() {
        return this.key;
}
```

La inicialización del identificador del recurso se realiza en el método initialize(). Para identificar univocamente a cada recurso se utiliza un recurso java que identifica a cada clase con un código (hashCode):

```
/* Initializes RPs and returns a unique identifier for this resource */
public Object initialize() throws Exception {
       this.key = new Integer(hashCode());

       // Initialize the resource properties

       return key;
}
```

Como método de enseñanza es suficiente con el método hashCode() de java pero no se garantiza este procedimiento para identificar de forma única a cada recurso. Si se realizan servicios más complejos se puede utilizar UUIDGen de Apache AXIS.

3.3.5 *Recurso Home*

En el anterior patrón de un único recurso el home se extiende de la clase de Globus SingletonResourceHome, ahora se va a extender de la clase ResourceHomeImpl de Globus que permite gestionar múltiples recursos.

```
package org.globus.examples.services.core.factory.impl;

import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.impl.ResourceHomeImpl;
import org.globus.wsrf.impl.SimpleResourceKey;

public class MathResourceHome extends ResourceHomeImpl {

       public ResourceKey create() throws Exception {
              // Create a resource and initialize it
              MathResource mathResource = (MathResource) createNewInstance();
              mathResource.initialize();
              // Get key
              ResourceKey key = new SimpleResourceKey(keyTypeName, mathResource
                            .getID());
              // Add the resource to the list of resources in this home
              add(key, mathResource);
              return key;
       }
}
```

Sólo se crea el método create que crea un nuevo recurso, lo inicializa y devuelve su identificador. El resto de métodos esperados están implementados en la clase padre ResourceHomeImpl.

3.3.6 *Descriptor de despliegue*

El nuevo fichero WSDD tiene que reflejar los dos nuevos servicios creados: el servicio factoría y el servicio instancia:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Instance service -->
    <service name="examples/core/factory/MathService" provider="Handler"
use="literal" style="document">
        <parameter name="className"
value="org.globus.examples.services.core.factory.impl.MathService"/>
```

```
<wsdlFile>share/schema/examples/MathService_instance/Math_service.wsdl</wsdlFile
>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
    </service>

    <!-- Factory service -->
  <service name="examples/core/factory/MathFactoryService" provider="Handler" use="literal"
style="document">
    <parameter name="className"
value="org.globus.examples.services.core.factory.impl.MathFactoryService"/>
    <wsdlFile>share/schema/examples/FactoryService/Factory_service.wsdl</wsdlFile>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="instance" value="examples/core/factory/MathService"/>
     </service>

</deployment>
```

Como en el patrón anterior se define dónde se sitúa el fichero wsdl y la clase del servicio factoría

Se introduce un nuevo parámetro que indica el nombre del servicio instancia.

El fichero jndi muestra los dos nuevos servicios necesarios:

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

<!-- Instance service -->
<service name="examples/core/factory/MathService">
      <resource name="home"
type="org.globus.examples.services.core.factory.impl.MathResourceHome">
      <resourceParams>

            <parameter>
                  <name>resourceClass</name>

      <value>org.globus.examples.services.core.factory.impl.MathResource</value
>
            </parameter>

            <parameter>
                  <name>resourceKeyType</name>
                  <value>java.lang.Integer</value>
            </parameter>

            <parameter>
                  <name>resourceKeyName</name>

      <value>{http://www.globus.org/namespaces/examples/core/MathService_instan
ce}MathResourceKey</value>
            </parameter>

            <parameter>
                  <name>factory</name>
                  <value>org.globus.wsrf.jndi.BeanFactory</value>
            </parameter>
```

```
      </resourceParams>
      </resource>
</service>

<!-- Factory service -->
<service name="examples/core/factory/MathFactoryService">
      <resourceLink name="home"
target="java:comp/env/services/examples/core/factory/MathService/home"/>
</service>

</jndiConfig>
```

El servicio factoría se enlaza con el servicio instancia mediante el tag resourceLink en donde se introduce el nombre del servicio.

De nuevo para deployar el servicio se genera el .gar , se para y se levanta de nuevo el contenedor para que tome los cambios realizados.

3.3.7 Cliente sencillo

El siguiente código muestra un cliente sencillo que accede al servicio desplegado anteriormente. Espera por línea de comandos la uri del servicio factoría:

```
package org.globus.examples.clients.FactoryService_Math;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import org.globus.examples.stubs.MathService_instance.GetValueRP;
import org.globus.examples.stubs.MathService_instance.MathPortType;
import
org.globus.examples.stubs.MathService_instance.service.MathServiceAddressingLoca
tor;
import
org.globus.examples.stubs.Factory.service.FactoryServiceAddressingLocator;
import org.globus.examples.stubs.Factory.FactoryPortType;
import org.globus.examples.stubs.Factory.CreateResource;
import org.globus.examples.stubs.Factory.CreateResourceResponse;

/* This client creates a new MathService instance through a FactoryService. This
client
 * expects one parameter: the factory URI.
 */
public class Client {

      public static void main(String[] args) {
            FactoryServiceAddressingLocator factoryLocator = new
FactoryServiceAddressingLocator();
            MathServiceAddressingLocator instanceLocator = new
MathServiceAddressingLocator();

            try {
                  String factoryURI = args[0];
                  EndpointReferenceType factoryEPR, instanceEPR;
                  FactoryPortType mathFactory;
                  MathPortType math;

                  factoryEPR = new EndpointReferenceType();
                  factoryEPR.setAddress(new Address(factoryURI));
                  mathFactory =
```

```
factoryLocator.getFactoryPortTypePort(factoryEPR);

                    CreateResourceResponse createResponse = mathFactory
                                .createResource(new CreateResource());
                    instanceEPR = createResponse.getEndpointReference();

                    math = instanceLocator.getMathPortTypePort(instanceEPR);

                    System.out.println("Created instance.");

                    // Perform an addition
                    math.add(10);

                    // Perform another addition
                    math.add(5);

                    // Access value
                    System.out
                                .println("Current value:" + math.getValueRP(new
GetValueRP()));

                    // Perform a subtraction
                    math.subtract(5);

                    // Access value
                    System.out
                                .println("Current value:" + math.getValueRP(new
GetValueRP()));
              } catch (Exception e) {
                    e.printStackTrace();
              }
       }

}
```

El cliente obtiene una referencia a la factoría portType y se usa para invocar la operación createResource. Se obtiene el objeto CreateResourceResponse a través del endpoint invcado antes. Con el ERP, se obtiene una referencia al PortType específico MathPortType que se usa para realizar las operaciones deseadas: add,..

3.4 Propiedades Recursos

Ejemplo de recursos en el fichero wsdl:

```
<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_inst
ance"

xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

       <!-- REQUESTS AND RESPONSES -->

       <!-- ... -->

       <!-- RESOURCE PROPERTIES -->

       <xsd:element name="Value" type="xsd:int"/>
       <xsd:element name="LastOp" type="xsd:string"/>
```

```
        <xsd:element name="MathResourceProperties">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
                </xsd:sequence>
        </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>
```

El elemento MathResourceProperties tiene dos elementos, el primero Value es de tipo int y el segundo LastOp es de tipo string.

Se ha realizado la declaración de los recursos en formato XML para de esta manera poder intercambiar información con otras entidades como servicios, clientes, etc.

Una declaración de tipo “unbounded” para un elemento representa la siguiente salida de una entidad:

```
<!-- RESOURCE PROPERTIES -->

<xsd:element name="Value" type="xsd:int"/>
<xsd:element name="LastOp" type="xsd:string"/>

<xsd:element name="MathResourceProperties">
<xsd:complexType>
       <xsd:sequence>
              <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="unbounded"/>
              <xsd:element ref="tns:LastOp" minOccurs="1"
maxOccurs="unbounded"/>
       </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

```
<MathResourceProperties
xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance">
       <tns:Value>10</tns:Value>
       <tns:Value>30</tns:Value>
       <tns:Value>50</tns:Value>
       <tns:Value>40</tns:Value>
       <tns:LastOp>ADDITION</tns:LastOp>
       <tns:LastOp>ADDITION</tns:LastOp>
       <tns:LastOp>ADDITION</tns:LastOp>
       <tns:LastOp>SUBTRACTION</tns:LastOp>
</MathResourceProperties>
```

Los interfaces estándar para la interacción con las propieades de los recursos son:

- **GetResourceProperty: Permite acceder a una propiedad recurso por su Qname.**
- GetMultipleResourceProperties: Permite acceder a varias propiedades recurso dado sus Qnames.
- SetResourceProperties:Permite actualizar, modificar o borrar Propiedades Recurso

- QueryResourceProperties: Permite consultas más complejas usando el lenguaje XPath en documentos Propiedades Recurso.

3.4.1 Accediendo a las propiedades recurso

Se toma como ejemplo de partida el servicio stateful para mostrar las propiedades de los recursos. Y se modifica la definición del wsdl:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"

targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_instance_rp"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
1.2-draft-01.wsdl"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />

<-- ... -->

<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty
          wsrpw:GetMultipleResourceProperties
          wsrpw:SetResourceProperties
          wsrpw:QueryResourceProperties"
    wsrp:ResourceProperties="tns:MathResourceProperties">

      <operation name="add">
            <input message="tns:AddInputMessage"/>
            <output message="tns:AddOutputMessage"/>
      </operation>

      <operation name="subtract">
            <input message="tns:SubtractInputMessage"/>
            <output message="tns:SubtractOutputMessage"/>
      </operation>

</portType>
</definitions>
```

Para obtener las propiedades de los recursos se definen 4 tipos de portTypes: GetResourceProperty, GetMultipleResourceProperties, SetResourceProperties, QueryResourceProperties.

Se define el namespaces:

```
http\://www.globus.org/namespaces/examples/core/MathService_instance_rp=
```

```
org.globus.examples.stubs.MathService_instance_rp
http\://www.globus.org/namespaces/examples/core/MathService_instance_rp/bindings
=

org.globus.examples.stubs.MathService_instance_rp.bindings
http\://www.globus.org/namespaces/examples/core/MathService_instance_rp/service=

org.globus.examples.stubs.MathService_instance_rp.service
```

No hay ningún cambio con los ficheros java a excepción de la no implementación del método getValueRP, debido a que no se usa en este ejemplo.

El fichero de despliegue wsdd se modifica para aceptar los nuevos portTypes:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="examples/core/rp/MathService" provider="Handler"
use="literal" style="document">
        <parameter name="className"
value="org.globus.examples.services.core.rp.impl.MathService"/>

<wsdlFile>share/schema/examples/MathService_instance_rp/Math_service.wsdl</wsdlF
ile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider GetMRPProvider
SetRPProvider QueryRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

Se despliega el .gar, el código del cliente llama a algún portType implementado anteriormente y la salida después de parar y arrancar el contenedor:

```
Value RP: 0
Value RP: 10
Value RP: 100
Value: 100
```

El siguiente código genera la parte correspondiente del total mostrado anteriormente llamando al método getResourceProperty:

```
printResourceProperties(math);
math.add(10);
printResourceProperties(math);
```

```
Value RP: 0
Value RP: 10
```

El método printResourceProperties contiene las siguientes instrucciones:

```
/*
```

```
 * This method prints out MathService's resource properties by using the
 * GetResourceProperty operation.
 */
private void printResourceProperties(MathPortType math) throws Exception {
      GetResourcePropertyResponse valueRP, lastLogRP;
      String value, lastLog;

      valueRP = math.getResourceProperty(MathQNames.RP_VALUE);

      value = valueRP.get_any()[0].getValue();

      System.out.println("Value RP: " + value);
}
```

Para actualizar propiedades utilizamos SetResourceProperties:

```
Value RP: 100
```

```
updateRP(endpoint, MathQNames.RP_VALUE, "100");
printResourceProperties(math);
```

```
/*
 * This method updates resource property "rpQName" in the WS-Resource
 * pointed at by the endpoint reference "epr" with the new value "value".
 */
private void updateRP(EndpointReferenceType epr, QName rpQName, String value)
            throws Exception {
      WSResourcePropertiesServiceAddressingLocator locator = new
WSResourcePropertiesServiceAddressingLocator();
      SetResourceProperties_PortType port = locator
                  .getSetResourcePropertiesPort(epr);

      UpdateType update = new UpdateType();
      MessageElement msg = new MessageElement(rpQName, value);
      update.set_any(new MessageElement[] { msg });

      SetResourceProperties_Element request = new
SetResourceProperties_Element();
      request.setUpdate(update);

      port.setResourceProperties(request);
}
```

Se utiliza la clase UpdateType para mandar un mensaje con la clase MessageElement y actualizar el elemento propiedad recurso.

Para obtener varios recursos se utiliza el método GetMultipleResourceProperties:

```
Value: 100
LastOp: ADDITION
```

Esta salida es generada por:

```
printMultipleResourceProperties(math);
/*
 * This method prints out MathService's resource properties by using the
 * GetMultipleResourceProperties operation.
 */
private void printMultipleResourceProperties(MathPortType math)
```

```
            throws Exception {
      GetMultipleResourceProperties_Element request;
      GetMultipleResourcePropertiesResponse response;

      QName[] resourceProperties = new QName[] { MathQNames.RP_VALUE,
                  MathQNames.RP_LASTOP };
      request = new GetMultipleResourceProperties_Element(resourceProperties);

      response = math.getMultipleResourceProperties(request);

      for(int i=0; i<response.get_any().length;i++)
      {
            String name = response.get_any()[i].getLocalName();
            String value = response.get_any()[i].getValue();
            System.out.println(name +": " + value);
      }
}
```

Se obtienen las propiedades múltiples a través del método getMultipleResourceProperties.

3.5 Gestión del ciclo de vida

Se presentan dos soluciones propuestas por la especificación del WSRF.

3.5.1 Destrucción inmediata

Es el más simple de las dos soluciones propuestas: permite hacer una petición del recurso que va a ser destruído de forma inmediata invocando al método destroy del servicio instancia.

Para realizar esta destrucción se tienen que seguir los siguientes pasos:

- Actualizar el fichero wsdl con el nuevo portType que extiende del estándar WSRF:

```
<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty
                    wsrlw:ImmediateResourceTermination"
    wsrp:ResourceProperties="tns:MathResourceProperties">

      <operation name="add">
            <input message="tns:AddInputMessage"/>
            <output message="tns:AddOutputMessage"/>
      </operation>

      <operation name="subtract">
            <input message="tns:SubtractInputMessage"/>
            <output message="tns:SubtractOutputMessage"/>
      </operation>

</portType>
```

- Se añaden las referencias introducidas en el fichero wsdl:

```
<definitions name="MathService"

targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_inst
ance_rl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance_r
l"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
    xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.wsdl"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-
01.wsdl"
    location="../../wsrf/lifetime/WS-ResourceLifetime.wsdl" />
```

o Se añaden parámetros al servicio:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Instance service -->
    <service name="examples/core/rl/MathService" provider="Handler"
use="literal" style="document">
        <parameter name="className"
value="org.globus.examples.services.core.rl.impl.MathService"/>

<wsdlFile>share/schema/examples/MathService_instance_rl/Math_service.wsdl</wsdlF
ile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider DestroyProvider"/>
    </service>

    <!-- Factory service -->
    <service name="examples/core/rl/MathFactoryService" provider="Handler"
use="literal" style="document">
        <parameter name="className"
value="org.globus.examples.services.core.rl.impl.MathFactoryService"/>

<wsdlFile>share/schema/examples/FactoryService/Factory_service.wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="instance" value="examples/core/rl/MathService"/>
    </service>

</deployment>
```

o Se crea el cliente de prueba:

```
package org.globus.examples.clients.FactoryService_Math_rl;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;
```

```
import org.globus.examples.stubs.MathService_instance_rl.MathPortType;
import
org.globus.examples.stubs.MathService_instance_rl.service.MathServiceAddressingL
ocator;
import
org.globus.examples.stubs.Factory.service.FactoryServiceAddressingLocator;
import org.globus.examples.stubs.Factory.FactoryPortType;
import org.globus.examples.stubs.Factory.CreateResource;
import org.globus.examples.stubs.Factory.CreateResourceResponse;

import org.oasis.wsrf.lifetime.Destroy;

/* This client creates a new MathService instance through a FactoryService. This
client
 * expects one parameter: the factory URI.
 */
public class Client_immed {

      public static void main(String[] args) {
            FactoryServiceAddressingLocator factoryLocator = new
FactoryServiceAddressingLocator();
            MathServiceAddressingLocator instanceLocator = new
MathServiceAddressingLocator();

            try {
                  String factoryURI = args[0];
                  EndpointReferenceType factoryEPR, instanceEPR;
                  FactoryPortType mathFactory;
                  MathPortType math;

                  // Get factory portType
                  factoryEPR = new EndpointReferenceType();
                  factoryEPR.setAddress(new Address(factoryURI));
                  mathFactory =
factoryLocator.getFactoryPortTypePort(factoryEPR);

                  // Create resource and get endpoint reference of WS-
Resource.
                  // This resource is our "instance".
                  CreateResourceResponse createResponse = mathFactory
                              .createResource(new CreateResource());
                  instanceEPR = createResponse.getEndpointReference();

                  // Get instance PortType
                  math = instanceLocator.getMathPortTypePort(instanceEPR);

                  System.out.println("Created instance.");

                  // Perform an addition
                  math.add(10);

                  // Perform another addition
                  math.add(5);

                  // Perform a subtraction
                  math.subtract(5);

                  math.destroy(new Destroy());
                  System.out.println("Destroyed instance.");
            } catch (Exception e) {
```

```
                    e.printStackTrace();
                }
        }
}
```

- Por supuesto se compila el cliente, se genera .gar y se para y levanta el contenedor de servicios web para que tome los cambios realizados.

Si después de destruir el portType se intenta acceder al servicio

```
math.destroy(new Destroy());
System.out.println("Destroyed instance.");

// Perform another addition
math.add(5);
```

Se muestra la siguiente excepción:

```
java.rmi.RemoteException: ; nested exception is:
        org.globus.wsrf.NoSuchResourceException
```

3.5.2 *Destrucción programada*

Este tipo de destrucción es un poco más compleja que la anterior. Permite definir exactamente cuándo se va a destruir el recurso.

Este tipo de destrucción está orientado a servicios con un tiempo de vida definido o bien recursos que pueden haberse congelado esperando alguna respuesta que nunca llega (fallo de red, errores de programación, etc).

El WSRF proporciona recursos para utilzarlos en la destrucción programada:

- TerminationTime: Especifica cuándo el rescurso debe ser destruido.
- CurrentTime: Tiempo de la máquina que alberga el recurso

3.5.3 *Fichero WSDL*

El portType tiene que extender de ScheduledResourceTermination:

```
<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty
                    wsrlw:ScheduledResourceTermination"
    wsrp:ResourceProperties="tns:MathResourceProperties">

       <operation name="add">
              <input message="tns:AddInputMessage"/>
              <output message="tns:AddOutputMessage"/>
       </operation>

       <operation name="subtract">
              <input message="tns:SubtractInputMessage"/>
              <output message="tns:SubtractOutputMessage"/>
       </operation>
</portType>
```

3.5.4 *Implementación del recurso*

La clase recurso tiene implementar la interfaz ResourceLifetime:

```
public class MathResource implements Resource, ResourceIdentifier,
```

```
            ResourceProperties, ResourceLifetime
```

Y crear una nueva propiedad de la clase para indicar el tiempo de finalización del recurso:

```
/* Resource properties */
private int value;
private String lastOp;
private Calendar terminationTime
```

Se añaden dos nuevas propiedades recursos:

```
/* Initializes RPs and returns a unique identifier for this resource */
public Object initialize() throws Exception {
      this.key = new Integer(hashCode());
      this.propSet = new SimpleResourcePropertySet(
                  MathQNames.RESOURCE_PROPERTIES);

            try {
            ResourceProperty valueRP = new ReflectionResourceProperty(
                        MathQNames.RP_VALUE, "Value", this);
            this.propSet.add(valueRP);
            setValue(0);

            ResourceProperty lastOpRP = new ReflectionResourceProperty(
                        MathQNames.RP_LASTOP, "LastOp", this);
            this.propSet.add(lastOpRP);
            setLastOp("NONE");

            ResourceProperty termTimeRP = new ReflectionResourceProperty(
                        SimpleResourcePropertyMetaData.TERMINATION_TIME, this);
            this.propSet.add(termTimeRP);

            ResourceProperty currTimeRP = new ReflectionResourceProperty(
                        SimpleResourcePropertyMetaData.CURRENT_TIME, this);
            this.propSet.add(currTimeRP);

      } catch (Exception e) {
            throw new RuntimeException(e.getMessage());
      }

      return key;
}
```

Se tiene que importar la siguiente clase para utilizar las propiedades TerminationTime y CurrentTime:

```
import org.globus.wsrf.impl.SimpleResourcePropertyMetaData;
```

Y finalmente se añaden los métodos obtener/establecer los tiempos de finalización:

```
/* Required by interface ResourceLifetime */
public Calendar getCurrentTime() {
      return Calendar.getInstance();
}

public Calendar getTerminationTime() {
      return this.terminationTime;
}
```

```
public void setTerminationTime(Calendar terminationTime) {
	this.terminationTime=terminationTime;
}
```

3.5.5 *Despligue*

Se necesita establecer un provider de Globus para el servicio de instancia:

```
parameter name="providers" value="GetRPProvider SetTerminationTimeProvider"/>
```

Y modificar el fichero jndi para indicar cada cuánto tiempo debe el contenedor observar la el tiempo de finalización:

```
<parameter>
	<name>sweeperDelay</name>
	<value>1000</value>
</parameter>
```

A continuación se genera el GAR.

3.5.6 *Cliente*

Se establece que a los 10 segundos se destruya el recurso

```
Calendar termination = Calendar.getInstance();
termination.add(Calendar.SECOND, 10);

SetTerminationTime request;
SetTerminationTimeResponse response;
request = new SetTerminationTime(termination);
response = math.setTerminationTime(request);

System.out.println("Current time               "
		+ response.getCurrentTime().getTime());
System.out.println("Requested termination time "
		+ termination.getTime());
System.out.println("Scheduled termination time "
		+ response.getNewTerminationTime().getTime());

boolean terminated = false;
int seconds = 0;
while (!terminated) {
	try {
		System.out.println("Second " + seconds);
		math.add(10);
		Thread.sleep(1000);
		seconds++;
	} catch (RemoteException e) {
		System.out.println("Resource has been destroyed");
		terminated = true;
	}
}
```

La ejecución del cliente genera la siguiente salida:

```
Created instance.
Current time               Sun Apr 03 00:54:29 CST 2005
Requested termination time Sun Apr 03 00:54:39 CST 2005
Scheduled termination time Sun Apr 03 00:54:39 CST 2005
Second 0
```

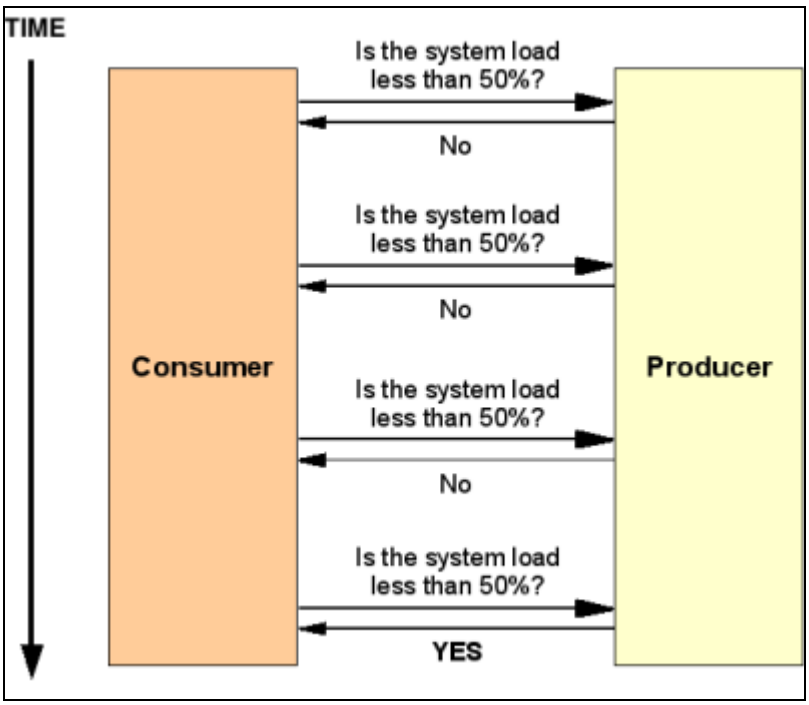
```
Second 1
Second 2
Second 3
Second 4
Second 5
Second 6
Second 7
Second 8
Second 9
Second 10
Resource has been destroyed
```

3.6 Notificaciones

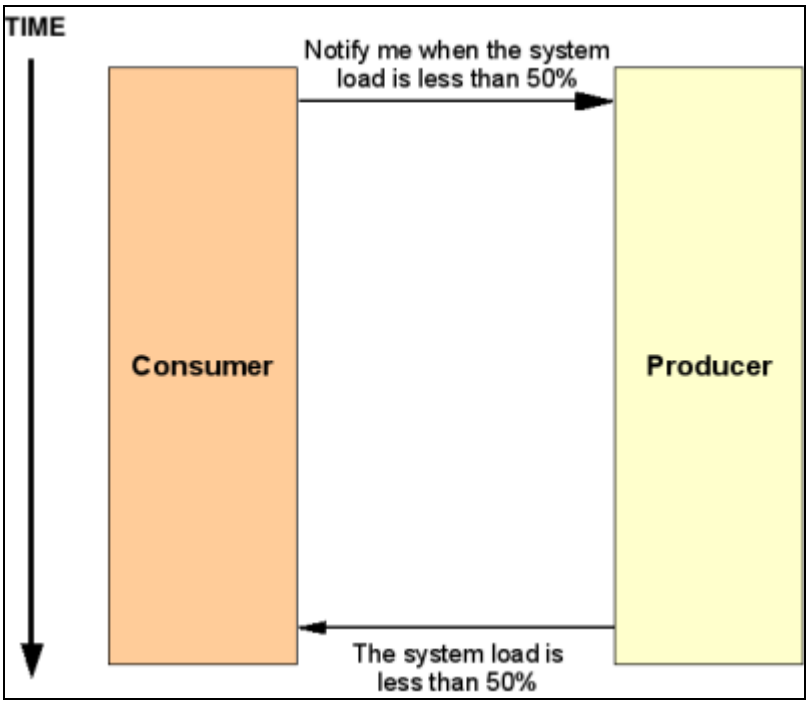
La notificación es un patrón de diseño que permite a los clientes ser informados cuando ocurre algún evento en el servidor. Dentro de Globus se utilizan las notificaciones-WS que permiten usar el diseño del patrón con servicios web sin tener que implementarlo desde cero.

En una aplicación existen diferentes componentes que dependiendo de los requisitos tendrán que estar sincronizados unos con otros, por ejemplo una aplicación con una GUI que tenga que mostrar en tiempo real información de base de datos tendrá que tener un sistema de notificaciones para que cuando se modifique la información en base de datos se modifique en la GUI.

La manera más directa para sincronizar estos componentes es mediante polling. El consumidor (GUI) pregunta constantemente al productor (BBDD) si se ha actualizado la información, así sucesivamente hasta que se informa afirmativamente, actualiza la GUI y sigue preguntando por si vuelve a cambiar. Esta solución no es eficiente considerando el tráfico de red y el tiempo de CPU consumido, y asi además existen más consumidores pueden llegar a saturar el productor.



Otra manera de implementar la sincronización es mediante la técnica de notificaciones, se manda un mensaje al productor para que mande una respuesta cuando se cumple lo pedido por el consumidor. El productor mantiene una lista de subscripciones de consumidores que esperan una notificación. De esta manera se evita cargar el tráfico de red con constantes mensajes preguntando por el estado del productor. Ejemplo:



3.6.1 Notificaciones-WS

Aunque las Notificaciones-WS no se encuentran dentro de las especificaciones del WSRF están fuertemente unidas a él. Proporciona un conjunto de interfaces estándar para usar el diseño del patrón de notificaciones con servicios web. Las Notificaciones-WS se componen de tres componentes:

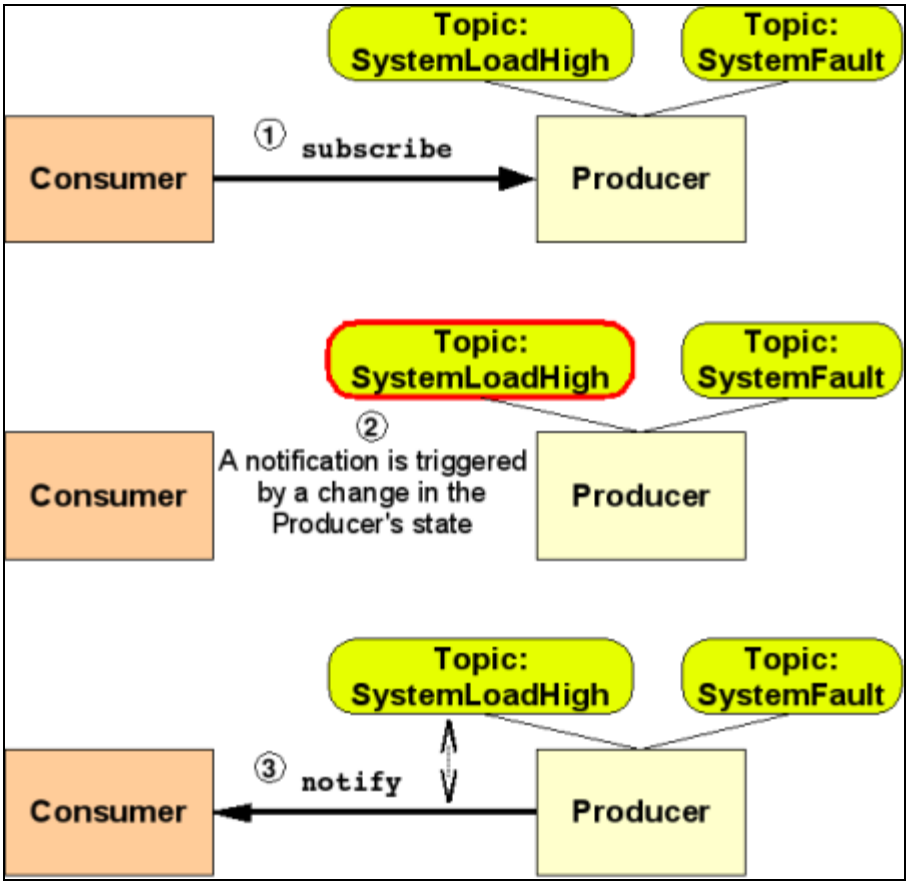
- WS-Topics

Los WS-Topics son asuntos, entidades las cuales con usadas para que los WS-Notificaciones se subscriban a ellos. Un servicio puede publicar un conjunto de tópicos que el cliente se ha subscrito. Se pueden además crear árboles de tópicos creando una jerarquía a la hora de notificar eventos, los eventos que se produzcan en un nodo se transmitirán al resto de hijos.

- WS-BaseNotification

Esta especificación define el estándar que tienen que cumplir tanto el productor como el consumidor. Esto quiere decir que el productor de notificaciones tiene que tener una operación para subscribir para que el consumidor de notificaciones pueda solicitar subscripciones. Sin embargo el consumidor de notificaciones tiene que tener una operación de notificación para que el productor de notificaciones pueda informar de nuevos eventos a los consumidores.

El siguiente gráfico muestra la interacción entre el consumidor y el productor:



Paso 1: El consumidor manda una operación de subscripción para notificar eventos del tópico: SystemLoadHigh. Internamente se ha creado un objeto en el productor con información de las subscripciones.

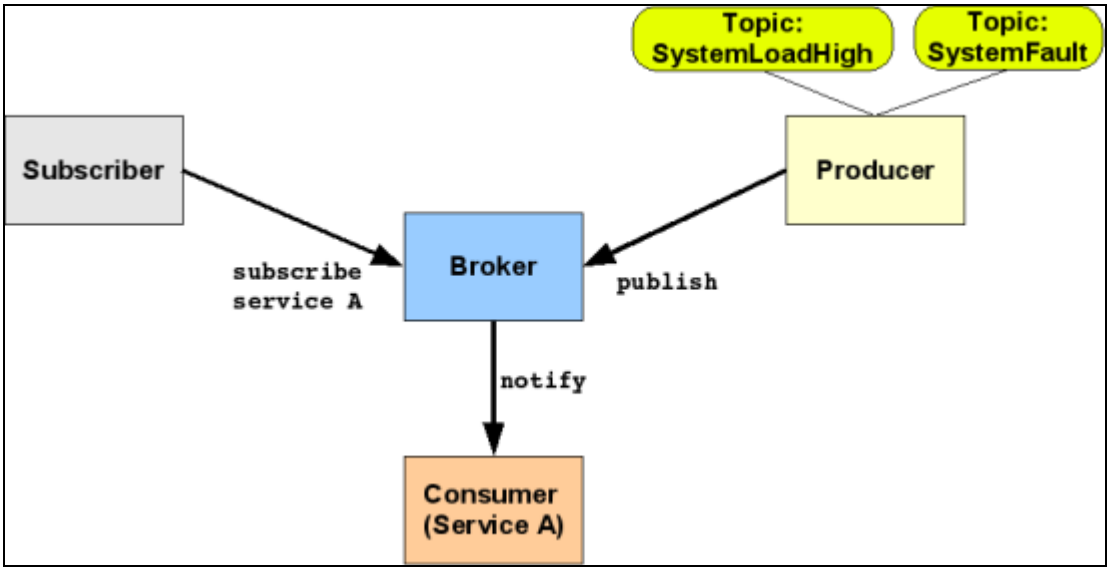
Paso 2: Se produce un evento del tópico SystemLoadHigh

Paso 3: El productor manda un mensaje de notificación a todos los subscriptores a este tópico.

o WS-BrokeredNotification

Con las notificaciones broker las notificaciones enviadas desde el productor a todos los subscriptores de un tópico se realiza mediante una nueva entidad intermediaria: el broker. El WS-BrokeredNotification define los estándares de los interfaces para la notifiación broker.

Ahora con el broker haciendo de intermediario entre el productor y el consumidor, el productor tiene que registrar y publicar todos los tópicos. El subscriptor, en este caso separado del consumidor, debe subscribirse a través del broker, no directamente con el productor. Cuando se produce una notificación, se notifica mediante el broker.



3.6.2 *Notificaciones con GT4*

GT4 no implementa complemetamente el estándar WS-Notificaciones, las notificaciones broker actualmente no son soportadas. Pero GT4 puede implementar sin esfuerzo el estándar WS-Notificaciones para que los valores de los RP (ResourcesProperties) sean considerados tópicos y así notificar eventos cuando cambien de valor.

3.6.3 *Notificando cambios en las propiedades recurso*

Se tomará como ejemplo el servicio web stateful desarrolla enteriormente para añadir notificaciones a un servicio para que los clientes sean informados cuando algún RP ha sido modificado.

3.6.3.1 El fichero WSDL

Se declara el nuevo namespaces y de importa el wsdl de las notificaciones:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"

targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_inst
ance_notif"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance_n
otif"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wsntw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-
01.wsdl"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
```

```
draft-01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />

<wsdl:import
  namespace=
  "http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
  location="../../wsrf/notification/WS-BaseN.wsdl"/>

    <!-- ... -->

</definitions>
```

El portType se extiende del NotificactionProducer:

```
<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty wsntw:NotificationProducer"
    wsrp:ResourceProperties="tns:MathResourceProperties">

  <!-- <operation>s -->
</portType>
```

Se crean los mappings entre los namespaces del fichero WSDL y los paquetes java:

```
http\://www.globus.org/namespaces/examples/core/MathService_instance_notif=

org.globus.examples.stubs.MathService_instance_notif
http\://www.globus.org/namespaces/examples/core/MathService_instance_notif/bindi
ngs=

org.globus.examples.stubs.MathService_instance_notif.bindings
http\://www.globus.org/namespaces/examples/core/MathService_instance_notif/servi
ce=

org.globus.examples.stubs.MathService_instance_notif.service
```

Nota: Siempre una línea por mapping.

3.6.3.2 Implementación del recurso

Anteriormente los atributos de la clase recurso se implementaban de la siguiente manera:

```
private int value;
private String lastOp;
```

Ahora se implementan de diferente manera:

```
/* Resource properties */
private ResourceProperty valueRP;
private ResourceProperty lastOpRP;
```

Se inicializan los recursos a través de la clase SimpleResourceProperty:

```
public MathService() throws RemoteException {
      this.propSet = new SimpleResourcePropertySet(
                  MathQNames.RESOURCE_PROPERTIES);

      try {
            valueRP = new SimpleResourceProperty(MathQNames.RP_VALUE);
            valueRP.add(new Integer(0));

            lastOpRP = new SimpleResourceProperty(MathQNames.RP_LASTOP);
```

```
                lastOpRP.add("NONE");
        } catch (Exception e) {
                throw new RuntimeException(e.getMessage());
        }

        this.propSet.add(valueRP);
        this.propSet.add(lastOpRP);
}
```

Para crear el recurso sólo es necesario las constantes QNames del recurso.

3.6.3.3 Publicación del tópico

Se utiliza la clase ResourcePropertyTopic para establecer como tópico un recurso.

Primero se implementa el interfaz TopicListAccessor, definiendo los metódos que extiende el interfaz (getTopicList):

```
import org.globus.wsrf.TopicListAccessor;

public class MathService implements Resource, ResourceProperties,
                TopicListAccessor {
        private TopicList topicList;

        // ...

        /* Required by interface TopicListAccessor */
        public TopicList getTopicList() {
                return topicList;
        }
}
```

Se inicializa la lista de tópicos, se crean los objetos ResourcePropertyTopic y se añaden a la lista de tópicos:

```
public MathService() throws RemoteException {
        /* Create RP set */
        this.propSet = new SimpleResourcePropertySet(
                        MathQNames.RESOURCE_PROPERTIES);

        /* Initialize the RP's */
        try {
                valueRP = new SimpleResourceProperty(MathQNames.RP_VALUE);
                valueRP.add(new Integer(0));

                lastOpRP = new SimpleResourceProperty(MathQNames.RP_LASTOP);
                lastOpRP.add("NONE");
        } catch (Exception e) {
                throw new RuntimeException(e.getMessage());
        }

        this.topicList = new SimpleTopicList(this);

        valueRP = new ResourcePropertyTopic(valueRP);
        ((ResourcePropertyTopic) valueRP).setSendOldValue(true);

        lastOpRP = new ResourcePropertyTopic(lastOpRP);
        ((ResourcePropertyTopic) lastOpRP).setSendOldValue(true);

        this.topicList.addTopic((Topic) valueRP);
        this.topicList.addTopic((Topic) lastOpRP);
```

```
        this.propSet.add(valueRP);
        this.propSet.add(lastOpRP);
}
```

3.6.3.4 Implementación del servicio

Se tienen que cambiar los métodos que forman las operaciones del servicio ya que para representar los recursos se utilizan clases SimpleResourceProperty. Por ahora no se han introducido cambios relacionado con las notificaciones ya que éstas se lanzan automáticamente.

Antes el método add se implementaba sin el sistema de notificaciones:

```
public AddResponse add(int a) throws RemoteException {
        value += a;
        lastOp = "ADDITION";

        return new AddResponse();
}
```

Ahora con las notificaciones el método que responde a la operación del servicio utiliza la clase ResourcePropertyTopic en vez de directamente los valores del recurso:

```
public AddResponse add(int a) throws RemoteException {
        Integer value = (Integer) valueRP.get(0);
        value = new Integer(value.intValue()+a);
        valueRP.set(0, value);
        lastOpRP.set(0,"ADDITION");

        return new AddResponse();
}
```

3.6.3.5 Descriptor de despliegue

Se añaden los proveedores de las subscripciones:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="examples/core/notifications/MathService" provider="Handler"
use="literal" style="document">
        <parameter name="className"
value="org.globus.examples.services.core.notifications.impl.MathService"/>

<wsdlFile>share/schema/examples/MathService_instance_notif/Math_service.wsdl</ws
dlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider SubscribeProvider
GetCurrentMessageProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>

</deployment>
```

A continuación se compilan los fuentes modificados y se genera el .gar

3.6.3.6 Código cliente

Para realizar la prueba de las notificaciones se crean dos clientes. Un cliente actuará de listener de notificaciones y el segundo cliente actuará de invocador de la operación add.

- Cliente Listener: Se compone de dos partes
  - Subscripción: Establece la subscripción con el valor del RP (Resource Property). Una vez que se establezca la subscripción, el bloque de este código no deja de ejecutarse.
  - Entrega (Delivery): El código de la entrega se ejecutará en cualquier momento que una notificación llegue al cliente. Dede implementar el interfaz `NotifyCallback`:

```
public class ValueListener implements NotifyCallback
```

El siguiente código implementa el método run que espera notificación del productor:

```
public void run(String serviceURI) {
	try {
		NotificationConsumerManager consumer;

		consumer = NotificationConsumerManager.getInstance();
		consumer.startListening();
		EndpointReferenceType consumerEPR = consumer
				.createNotificationConsumer(this);

		Subscribe request = new Subscribe();
		request.setUseNotify(Boolean.TRUE);
		request.setConsumerReference(consumerEPR);

		TopicExpressionType topicExpression = new TopicExpressionType();
		topicExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
		topicExpression.setValue(MathQNames.RP_VALUE);
		request.setTopicExpression(topicExpression);

		WSBaseNotificationServiceAddressingLocator notifLocator =
			new WSBaseNotificationServiceAddressingLocator();
		EndpointReferenceType endpoint = new EndpointReferenceType();
		endpoint.setAddress(new Address(serviceURI));
		NotificationProducer producerPort = notifLocator
				.getNotificationProducerPort(endpoint);

		producerPort.subscribe(request);

		System.out.println("Waiting for notification. Ctrl-C to end.");
		while (true) {
			try {
				Thread.sleep(30000);
			} catch (Exception e) {
				System.out.println("Interrupted while sleeping.");
			}
		}
	} catch (Exception e) {
		e.printStackTrace();
	}
}
```

Se implementa en el cliente el sistema de notificación del consumidor. Habilitará un método Notify que será invocado por el productor. El cliente generado actuará las veces de servidor y de cliente gracias a la clase Globus NotificationConsumerManager que permite esta funcionalidad. Después de arrancar el listener de peticiones se realiza la subscripción. Al final se crea un bucle esperando para recibir notificaciones.

El código que implementa la entrada de las notificaciones sería el siguiente, tiene que implementar el método delivery ya que el servicio implementa el interfaz NotifyCallback:

```
public void delivery(List topicPath, EndpointReferenceType producer,
              Object message) {
       ResourcePropertyValueChangeNotificationElementType notif_elem;
       ResourcePropertyValueChangeNotificationType notif;

       notif_elem = (ResourcePropertyValueChangeNotificationElementType)
message;
       notif = notif_elem.getResourcePropertyValueChangeNotification();

       if (notif != null) {
              System.out.println("A notification has been delivered");


              System.out.print("Old value: ");
              System.out.println(notif.getOldValue().get_any()[0].getValue());
              System.out.print("New value: ");
              System.out.println(notif.getNewValue().get_any()[0].getValue());
       }
}
```

El método delivery tiene los siguientes parámetros:

- topicPath: El tópico que ha producido la notificación.
- producer: El productor de la notificación.
- message: Notificación actual. Se tendrá que hacer un casting al recibirlo como Object.

Si la notificación llega correctamente se pinta el valor antoguo y el nuevo.

A continuación para poder ver en ejecución el ejemplo se compila el código fuente modificado del listener, se ejecuta el cliente y se muestra el sisguiente mensaje en espera:

```
Waiting for notification. Ctrl-C to end.
```

Luego se compila el código del cliente que ejecuta la operación add y se lanza a ejecución:

```
Value RP: 10
LastOp RP: ADDITION
```

La consola del listener debería mostrar el siguiente mensaje:

```
A notification has been delivered
Old value:0
New value:10
```

De nuevo en el cliente del add:

```
Value RP: 20
LastOp RP: ADDITION
```

Y en la consola del listener:

```
A notification has been delivered
Old value:10
New value:20
```

4. Servicios Web en GT4

Un servicio web es un sistema diseñado para soportar la interoperabilidad entre dos sistemas software en una arquitectura distribuida. El interfaz que expone está descrito según el estándar de servicios web en un fichero según la especificación WSDL. Inicialmente se concibió con la idea de desacoplar servicios ofreciendo tanto muna mayor escalabilidad como una potencial capacidad de integración entre organizaciones, conceptos que se mantienen en la propuesta de valor actual pero que han perdido pretenciosidad, orientándose a soluciones localizadas y mucho más específicas.

Los servicios web se encargan, por tanto, de estandarizar los mensajes que dos sistemas usarán para comunicarse, haciéndolo, generalmente, sobre el protocolo HTTP, codificando los mensajes según SOAP y utilizando descriptores a partir de WSDL.

Una de las virtudes de los servicios web es la capacidad de integración con otros estándares que permiten agregar funcionalidad a los sistemas. Según de qué tecnología, estos servicios son bastante diferentes pero se suelen basar en la infraestructura y en funcionalidades verticales: persistencia, seguridad, gestión de logs, etcétera.

Dado que se trata de un concepto todavía en crecimiento al que le queda mucho camino por recorrer para poder hablar de su madurez y sobre el que se definen y redefinen multitud de estándares y especificaciones, en GT4 se han incorporado algunas de ellas de manera casi definitiva pero no podemos hablar de que se haya completado la incorporación de otras nuevas.

Las más relevantes se detallan a continuación.

4.1 OGSA

Desde la aparición de la versión 3 de Globus Toolkit, la arquitectura ha incorporado tanto la filosofía de servicios web como la orientación a servicios de SOA (Anteriormente conocido como SOAP, pero que ha evolucionado hacia un concepto más amplio) que se traduce en Open Globus Service Architecture, OGSA.

OGSA define la arquitectura estándar de lo que debe ser una aplicación basada en grid a partir de la definición de interfaces que se encargan de establecer:

- Cómo se establece la identidad y se negocia la autenticación.
- Cómo son expresadas las políticas.
- Cómo se descubren servicios.
- Cómo se gestionan las comunicaciones y los servicios.
- …

4.2 WSRF

Para cumplir con los requisitos de las aplicaciones basadas en grid no basta con OGSA. Dado que una de las características implícitas de la invocación de un servicio es el estado, no basta con un servicio web, sino que es necesario que exista infraestructura que permita conservar el estado de dicho servicio, contradiciendo el concepto de integración inicial. Para ello se seleccionó la especificación de OASIS Web Services Resource Framework que hace posible la

cuadratura del círculo en torno a los servicios web.

GT4 Incluye una implementación de la especificación WSRF como parte de su infraestructura, que conforma el núcleo y permite el desarrollo de aplicaciones siguiendo uno de los principios fundamentales del grid: la gestión de recursos.

4.3 WS-Addressing

La especificación WS-Addressing define mecanismos de direccionamiento de mensajes y servicios web neutrales con respecto al transporte. Concretamente, esta especificación define elementos XML que identifican la ubicación de servicios web y que aseguran la dirección de los mismos en los mensajes utilizados en una operación de comunicación entre dos puntos.

4.4 WS-Notification

Otro de los estándares interesantes a propósito de los servicios web se refiere a los mecanismos de publicación-suscripción (publish-subscribe). WS-Notification es una familia de especificaciones que define una aproximación basada en el patrón publish-subscribe que permite disponer de varios sistemas de notificación: WS-Notification, WS-Topics y WS-BrokeredNotification. Naturalmente, la notificación se apoya en WS-Addressing y WSRF para ofrecer cualquier servicio requerido.

En GT4 se torna especialmente útil en tareas de gestión de Jobs.

4.5 WS-Security y SAML

La seguridad es una de las características más importantes en GT4. Para ello incorpora la familia de especificaciones de seguridad basada en servicios web y el Security Authorization Markup Language (SAML), principalmente para la protección de mensajes, autenticación, delegación y atorización:

- Transport Level Security o WS-Security y WS-SecureConversation, a nivel de mensaje, son usados como mecanismos de protección de mensajes en cominación con SOAP.
- Certificados de entidad X.509 o la combinación de usuario y clave se usan como credenciales de autenticación.
- Certificados Proxy X.509 y WS-Trust se usan para la delegación.
- SAML sirven para la autorización

Más adelante se detallará el sistema de seguridad de GT4, Globus Security Infrastructure (GSI).

4.6 WSDM

Otro de os estándares que han nacido del comité técnico de OASIS es el de Web Services Distributed Management. Se trata de intentar resolver el problema de la gestión de diferentes servicios de forma unificada en lugar de tener un conjunto heterogéneo de sistemas de gestión.

WSDM aporta significativo valor en tres grupos:

- Clientes con sistemas de información heterogéneos. WSDM permite gestionar software de diferentes fabricantes.
- Para ISVs WSDM provee estándares de identificación, inspección y modificación de características de recursos.
- Fabricantes de dispositivos: WSDM permite exponer interfaces de gestión utilizando servicios web de manera estándar, sin tener en cuenta la manera en la que esos dispositivos han sido construidos.

WSDM consta de dos especificaciones.

Management Using Web Services (MUWS) define cómo se representa y accede al interface de gestión como servicio web. Crea una capa de integración entre la gestión y los diferentes protocolos usados en los recursos.

Management Of Web Services (MOWS) plasma cómo se gestiona un servicio web como recurso y cómo se describe y accede a dicha gestión usando MUWS. Provee mecanismos y métodos que permiten la gestión de servicios web para operar más allá de los límites de la organización.

Depende de las otras especificaciones: WSRF, WS-I, WS-N y WS-A.

4.7 WS-Interoperability

Se trata de una organización de perfiles que definen cómo las especificaciones existentes a cerca de servicios web pueden ser usadas para promover la interoperabilidad a través de diferentes implementaciones. Por ejemplo, el perfil WS-I BASIC se refiere a la descripción de mensajes y servicios, sobre todo XML, SOAP y WSDL; WS-I BASIC SECURITY añade mecanismos de seguridad.

Evidentemente, WS-I siempre se encuentra en fase de desarrollo y evolución continua puesto que su completitud no tiene sentido.

5. Globus Security Infrastructure (GSI)

5.1 Descrición general

GSI utiliza criptografía de clave pública (también conocido como criptografía asimétrica) como la base de toda su funcionalidad. De hecho, los términos y conceptos usados en GSI proceden de la criptografía de clave pública.

Las principales motivaciones que subyacen bajo la infraestructura de seguridad de globus son:

- La necesidad de una comunicación segura, autenticada y quizá confidencial, entre los elementos de computación del grid.
- La exigencia de soportar sistemas de seguridad más allá de los límites de la organización.
- Satisfacer el requisito de disponer de mecanismos de single sign-on para los usuarios del grid, incluyendo la delegación de credenciales para computaciones que requieran de más de un recurso.

5.2 Conceptos clave

5.2.1 Criptografía de clave pública

Lo más relevante que concierne a la criptografía de clave pública, en contraposición a los sistemas de criptografía clásicos, es que no utiliza sólo una clave, contraseña o código secreto, sino que usa dos. Estas claves son números están relacionados matemáticamente tal que si la clave es usada para encriptar un mensaje, la otra clave debe servir para desencriptarlo. Tambíen es importante el hecho de que es prácticamente imposible, teniendo en cuenta el avance actual de la capacidad de cálculo de los sistemas de información, obtener una segunda clave a partir de la primera y desencriptar cualquier mensaje.

Haciendo pública una de estas dos claves (a la sazón, la clave pública) y manteniendo la otra privada, una persona puede probar si se trata de la clave privada encriptando un mensaje cualquiera. Si el mensaje puede ser desencriptado con la clave pública, entonces se puede decir con certeza que se posee la clave privada.

Es importante, por tanto, que las claves privadas no sean conocidas y se mantengan secretas ya que, en caso contrario, perderían el valor de privacidad con respecto al propietario.

5.2.2 Firma digital

Usando criptografía de clave pública es posible firmar digitalmente una porción de información. Firmar significa, esencialmente, asegurar al receptor de la información que ese mensaje no ha sido forzado en el camino antes de llegar a sus manos.

Para formar un mensaje se necesita obtener un identificador único, para lo que se calcula un hash de la información (este algoritmo debe ser conocido por las dos partes). Usando la clave privada se encripta el hash, y se adjunta al mensaje, asegurándose de que el receptor tiene la clave pública.

Para verificar que el mensaje firmado es auténtico, el receptor del mensaje calculará el hash del mensaje usando el mismo algoritmo para luego desencriptar el hash enviado y compararlo con el generado. En caso de que sean iguales, entonces está claro que el mensaje no ha sido cambiado desde que fue firmado.

5.2.3 Certificados

Otro de los conceptos clave de la autenticación GSI es la de los certificados. Todos los usuarios y servicios en el grid son identificados mediante un certificado que contiene información vital para realizar dicha identificación y la autenticación del usuario o del servicio.

Un certificado GSI incluye cuatro piezas de información:

- Un nombre que identifique a la persona u objeto al que el certificado representa.
- La clave pública que pertenece al sujeto anterior.
- La identidad de l autoridad certificadora (Certificate Authority, CA) que ha firmado el certificado para certificar que la clave pública y la identidad son correctas.
- La forma digital de la autoridad certificadora.

Es muy relevante que es necesaria la participación de un tercero para certificar que el vínculo entre la clave pública y la persona u objeto/servicio son las que están en el certificado. Para que todo esto tenga sentido, la autoridada certificadora no puede ser cualquier tercero, sino que debe tratarse de una entidad de confianza (existen organismos que regulan qué entidades son de confianza en Internet) ya que la comunicación entre el cliente y la entidad no puede ser establecida mediante comunicaciones securizadas.

Los certificados en GSI están codificados según el estándar para certificados X.509, establecido por el Internet Engineering Task Force (IETF). Estos certificados pueden ser compartidos por otro software así como los navegadores (browsers) más populares.

5.2.4 Autenticación mutua

Si las dos partes tienen un certificado y ambas 'confían' en las autoridades certificadoras que firmaron ambos certificados, entonces dichas partes pueden probar que son quienes dicen que son. Esto también es conocido como autenticación mutua. GSI utiliza SSL (Secure Socket Layer, también conocido por el estándar de el IETF Transoport Level Security, TLS) para soportar el mecanismo de autenticación mutua.

Antes de que la autenticación mutua suceda, las partes envueltas deben confirmar que la autoridad certificadora que firmó los certificados es de confianza. En la práctica, esto significa que cada parte tiene una copia del certificado de la autoridad certificadora que contiene la clave privada de la entidad, así que lo que hace cada parte es confiar en que ese certificado pertenece realmente a la autoridad certificadora indicada.

Para realizar una autenticación mutua, una persona A, por ejemplo, establece una conexión con otra persona B. Entonces comienza el proceso: A envía el certificado a B.

El certificado le dice a B quién dice A que es ´(su identidad), cuál es la clave pública de A y qué autoridad certificadora está siendo usada para el certificado de A.

Primero B se tiene que asegurar de que el certificado es válido verificando la firma digital de la autoridad certificadora, chequeando que la entidad firmó el certificado y que el certificado no ha sido forzado.

Una vez que B se ha cerciorado de que el certificado de A es válido, debe asegurarse de que A es la persona identificada por el certificado enviado. Para ello B genera un mensaje aleatorio y se lo envía a A para que lo encripte. A encripta el mensaje usando su clave privada y se lo envía a B para que lo desencripte. B usa la clave pública y verifica si se trata del mensaje original. En caso afirmativo, B tiene por seguro que A es quien dice ser que es.

Ahora que B sabe quién es A, A debe asegurarse de la identidad de B, comenzando el proceso en sentido inverso.

Si ambos procesos concluyen con éxito, A y B han establecido una conexión en la que están seguros de cuál es la identidad del otro.

5.2.5 Comunicación confidencial

Por defecto, GSI no establece comunicaciones confidenciales (encriptadas) con otras partes. Una vez que la autenticación mutua se ha realizado, GSI deja de participar en la comunicación para no añadir peso a la encriptación/desencriptación de las operaciones y mensajes enviados durante el proceso.

GSI puede ser utilizado fácilmente para establecer una clave compartida para la encriptación si se requieren comunicaciones confidenciales.

Otra de las características importantes en cuanto a confidencialidad es la integridad en la comunicación. Integridad significa que un tercero podría interceptar los mensajes pero no modificarlos de ninguna manera. GSI provee de mecanismos de integridad por defecto, aunque pueden ser desactivados, ya que introducen cierta carga en las comunicaciones, pero no con tanto peso como la encriptación.

5.2.6 Clave pública segura

El núcleo de GSI de GT4 espera que la clave privada del usuario esté guardada en el sistema de ficheros del sistema local. Para prevenir que otros usuarios del sistema puedan obtener la clave privada, el fichero que la contiene está encriptado con una clave (passphrase). Para usar GSI, el usuario debe introducir esa clave requerida para desencriptar el fichero que contiene la clave privada.

Existe además un prototipo para el uso de smartcards en GSI que permitiría al usuario almacenar su clave privada en la memoria de la tarjeta en lugar de en un fichero, haciendo más difícil su uso fraudulento y la usurpación de identidad.

5.2.7 Delegación, Single Sign-on y Certificados Proxy

GSI dispone de características de delegación como extensión del estándar SSL que reduce el número de veces que el usuario debe introducir su clave (passphrase). Si un grid requiere de varios recursos para ser utilizado y cada uno de ellos requiere autenticación mutua o si otro agente local o remoto realiza una petición a un servicio en nombre de un usuario, la necesidad de introducir la clave puede ser evitada mediante la creación de un proxy.

En este contexto, un proxy consiste en un nuevo certificado y una clave privada. El nuevo certificado es firmado por el propietario en lugar de una autoridad certificadora. Además, indica que se trata de un proxy y una fecha de caducidad que limita la vida del certificado proxy.

La clave privada del proxy debe mantenerse en un lugar seguro porque, aunque el proxy no

tenga una validez muy prolongada, puede suponer el mismo riesgo que la propia clave del usuario.

5.3 Descriptores de seguridad

Los descriptores de seguridad contienen varias propiedades y credenciales como la ubicación del fichero grid-mapfile, requerido para los mecanismos de autenticación y autorización. Existen cuatro tipos de descriptores de seguridad en la instalación de GT4: el del contenedor, el del servicio, el del recurso y el del cliente. De los tres primeros se determina una jerarquía en la que el nivel de seguridad es heredado y sobrescrito por los niveles más bajos si fuese necesario para reforzar el nivel de seguridad si fuese necesario.

Estas alteraciones se pueden realizar de dos formas: en el código del servicio o recurso, o de forma declarativa, en los descriptores XML asociados. En el segundo caso es importante conocer que éstos descriptores no son cargados dinámicamente y que cualquier cambio no se tornará efectivo instantáneamente (no hay cambios en caliente) , debiendo volver a iniciar el contenedor.

5.3.1 Server-side

5.3.1.1 Credenciales

El contenedor y cada servicio deben ser configurados de forma independiente. Las credenciales pueden ser fijadas de dos formas:

- Indicando la ruta al fichero proxy.
- Indicando la ruta al certificado y al fichero de claves.

En este caso, el fichero de credenciales sí que tendrá efecto inmediato en caso de ser modificado.

```
<securityConfig xmlns="http://www.globus.org">
   ...<credential><key-file value="keyFile"/>
      <cert-file value="certFile"/>
   </credential>
   ...
</securityConfig>
```

O bien:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <proxy-file value="proxyFile"/>
   ...
</securityConfig>
```

El framework buscará las credenciales siguiendo el orden de más local a más general. Es decir, comenzará por las credenciales del recurso, seguirá con las del servicio, luego con las del contenedor y luego tomará las credenciales por defecto.

5.3.1.2 *Configurando grid-mapfile*

El contenedor y cada servicio pueden ser configurados para usar un fichero grid-mapfile diferente. Para ello existe una directiva en el descriptor en la que indicar la ubicación de dicho fichero.

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <gridmap value="ubicación gridMapFile"/>
   ...
```

```
</securityConfig>
```

El mismo orden que en el caso de las credenciales se usará para para realizar la autenticación utilizando grid-mapfile.

5.3.1.3 *Configurando los métodos de autenticación*

Los métodos de autenticación sólo tienen sentido a nivel de servicios.

El método de autenticación de un servicio es indicado con el elemento XML `<auth-method/>`.

La granularidad permitida en el descriptor se reduce a dos ámbitos: global y por método, como se verá a continuación.

Los métodos que se soportan hasta la fecha son:

- `<none/>`. Indica que no es necesaria la realización de ninguna autenticación por parte del cliente que accede al servicio. Este método no es combinable con ningún otro método de autenticación ya que carecería de sentido.
- `<GSISecureMessage/>`. Indica que el método de securización de mensajes de GSI será empleado. El subelemento <protection-level/> puede también ser aplicado según las siguientes opciones:
    a. `<integrity/>`. Indica que el mensaje tiene que estar protegido para garantizar su integridad, es decir. Firmado.
    b. `<privacy/>`. Indica que el mensaje debe estar firmado y encriptado.
- <GSISecureConversation/>. Indica que el método de securización de conversación de GSI será el utilizado según las siguientes opciones:
    a. `<integrity/>`. Indica que el mensaje tiene que estar protegido para garantizar su integridad, es decir. Firmado.
    b. `<privacy/>`. Indica que el mensaje debe estar firmado y encriptado.
- `<GSITransport/>`. Indica que el método de securización de trasnporte de GSI será el utilizado según las siguientes opciones:
    a. `<integrity/>`. Indica que el mensaje tiene que estar protegido para garantizar su integridad, es decir. Firmado.
    b. `<privacy/>`. Indica que el mensaje debe estar firmado y encriptado.

A continuación se muestra un ejemplo de un descriptor de seguridad tipo:

```
<securityConfig xmlns="http://www.globus.org">

   <method name="findServiceData">
      <auth-method>
         <none/>
      </auth-method>
   </method>

   <method name="destroy">
      <auth-method>
         <GSISecureMessage/>
         <GSISecureConversation>
            <protection-level>
               <integrity/>
```

```
            </protection-level>
         </GSISecureConversation>
      </auth-method>
   </method>

   <!—método de autenticación para cualquier otro método -->
   <auth-method>
      <GSISecureConversation/>
   </auth-method>
</securityConfig>
```

5.3.1.4 Configurando el modo run-as

El elemento de la configuración <run-as> es usado para configurar la identidad con la que se ejecutará el servicio. Esto también puede ser configurado tanto a nivel de método como a nivel general para cada servicio, usando los siguientes parámetros de configuración:

- `<caller-identity/>`. El método del servicio será ejecutado con la identidad del cliente. Podrá contentener lo siguiente:
  a. Si se usa GSISecureMessage: el firmante de la identidad será añadido al conjunto de identidades. También, la cadena de firmantes del certificado será incluida en las credenciales del objeto.
  b. Si se usa GSISecureConversation: la identidad del que inicó la ejecución será añadido al conjunto de indetidades. Si se realizó la autenticación del cliente, la cadena de certificados será también añadida al conjunto de credenciales públicas. Si, además, se realizó delegación, las credenciales delegadas se añadirán al conjunto de credenciales privadas.
  c. Si se usa autorización basada en grid-mapfile, se añadirá el UserNamePrincipal.
- `<system-identity/>`. El método de servicio se ejecutará con la identidad del contenedor.
- `<service-identity/>`. El método de servicio se ejecutará con la identidad del propio servicio, en caso de que tenga uno. En caso contrario se usará el del contenedor.
- `<resource-identity/>`. El método de servicio se ejecutará con la identidad del recurso en caso de que tenga. En caso contrario se utilizará el del servicio o el del contenedor en último término.

A continuación se muestra un ejemplo:

```
<securityConfig xmlns="http://www.globus.org">
   <method name="add">
      <run-as>
         <caller-identity/>
      </run-as>
   </method>

   <method name="subtract">
      <run-as>
         <system-identity/>
      </run-as>
   </method>

   <!-- default run-as for any other method -->
   <run-as>
      <service-identity/>
   </run-as>
```

```
</securityConfig>
```

5.3.1.5 Configurando los mecanismos de autorización

El contenedor y cada servicio puede ser configurado con una cadena de autorización (ver patrón Chain of responsibility), mecanismo también conocido como Política de Puntos de Decisión (Polocy decisión Points, PDPs), usando el elemento `<authz/>`. Cada PDP tiene un prefijo y un nombre de dominio cualificado, FQDN. El prefijo es usado para permitir que existan mútliples instancias del mismo PDP en la cadena de autorización. Por ejemplo:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <authz
value="self:org.globus.wsrf.impl.security.authorization.SelfAuthorization"/>
   ...
<securityConfig/>
```

Cada PDP es instanciado con información de configuración que puede ser obtenida para tener más información de lo que el PDP pueda necesitar para tomar decisiones de autorización. Si la cadena de autorización es configurada a nivel de contenedor, entonces los parámetros son obtenidos de la configuración global a partir del descriptor de despliegue del contenedor. En caso de estar configurado a nivel de servicio, el PDP obtendrá los parámetros del descriptor de despliegue del servicio. En caso de ser necesario especificar la configuración o parte de ella a nivel de recurso, esto sólo podrá realizarse a través del código, nunca en la configuración.

Los siguientes PDPs son partes de GT4 y pueden ser utilizados *out-of-the-box*:

none	No se realiza ninguna autorización.
self	Nombre PDP: selfAuthz:org.globus.wsrf.impl.security.authorization.SelfAuthorization No requiere ningún parámetro de configuración.
gridmap	Nombre PDP: grimapAuthz:org.globus.wsrf.impl.security.authorization.GridMapAuthorization Se realiza la autorización según el fichero grid-mapfile.
identity	Nombre PDP: idenAuthz:org.globus.wsrf.impl.security.authorization.IdentityAuthorization Espera que la identidad sobre la que ejecutar la autorización sea configurada a nivel de servicio o contenedor La identidad del cliente debe coincidir con dicha propiedad.
host	Nombre PDP: hostAuthz:org.globus.wsrf.impl.security.authorization.HostAuthorization La propiedad hostAuthz-url debe ser fijada ya que será la que se verifique cuando se haga una petición.
userName	Nombre PDP: userNameAuthz:org.globus.wsrf.impl.security.authorization.UsernameAuthori

	zation No require configuración adicional Esto utiliza el modulo de control de acceso de JAAS (Java Authentication and Authorization Service) basado en pares de usuario y clave. This uses the configured JAAS Login Module to authorize the user based on username and password. The PDP uses NameCallback and PasswordCallback to send user name and password information to the Login module.
samlCallout	Nombre PDP: samlAuthz:org.globus.wsrf.impl.security.authorization.SAMLAuthorizationCallout Cuando se especifica este PDP a través del alias samlCallout alias o con la constante org.globus.wsrf.impl.security.authorization.Authorization.AUTHZ_SAML se utiliza el perfijo samlAuthz para obtener la configuración. La llamada para realizar una autorización con SAML PDP se configurará con los parámetros de la talba siguiente.

<prefix>-authzService	La URL del servicio de autorización.
<prefix>-authzServiceIdentity	La identidad a usar en la autorización del servicio. Si no se especifica ninguna identidad entonces se realiza la llamada como si se tratase de una llamada con self-authorization.
<prefix>-authzServiceCertificateFile	Fichero que identifica el certificado a usar cuando se encripten los mensajes y se verifiquen las respuestas firmadas a nivel SAML. Esto solo es necesario si se usa GSISecureMessage con protección de privacidad o si el usuario pide la firma SAML.
<prefix>-authzServiceCertificate	Este parámetro es equivalente al anterior, pero solo puede usarse a través del código, no en la configuración. Debe ser fijado con un valor del tipo java.security.cert.X509Certificate.
<prefix>-securityMechanism	Se trata del mecanismod e seguridad a emplear. Los valores permitidos son: none, msg (GSISecureMessage) y conv (GSISecureConversation). La seguridad a nievl de transporte debe ser indicada especificando URLs HTTPS en el prefijo.
<prefix>-protectionLevel	Indica el nivel de protección. Los valores válidos son sig (protección de integridad) y enc (protección de privacidad e integridad). Por defecto vale sig.
<prefix>-samlAuthzReqSigned	Determina si la petición está firmada internamente o no. Las peticiones SAML pueden incluir una firma en la petición y en la respuesta. Esto está separado de cualquier mecanismo de seguridad aplicado incluso a nivel de SOAP o de transporte. Los valores son trae o false, este último es el valor por defecto.

<prefix>-samlAuthzSimpleDecision	Determina si una petición es una decisión simpre o no. Los valores válidos son true y false.

Cualquier otro esquema puede ser configurado y añadido a los anteriores, pudiendo implementar un sistema de autorización propio.

5.3.2 Ejemplo de sistema de autorización propio

Dentro de la API GT4 y de la infraestructura WSRF existe un interfaz que cualquier sistema de autorización a medida debe implementar para integrarse correctamente. He aquí un ejemplo:

```
public class MiPDP implements org.globus.wsrf.security.PDP {

    private Principal authorizedIdentity;

    /* No usado en el ejemplo */
    public String[] getPolicyNames() {
        return new String[0];
    }

    /* No usado en el ejemplo */
    public Node getPolicy(Node query) throws InvalidPolicyException {
        return null;
    }

    /* No usado en el ejemplo */
    public Node setPolicy(Node policy) throws InvalidPolicyException {
        return null;
    }

    public boolean isPermitted(Subject peerSubject,
                               MessageContext context, QName operation)
        throws AuthorizationException {

        if (peerSubject == null)
            return false;

        Set peerPrincipals = peerSubject.getPrincipals();

        if ((peerPrincipals == null) || peerPrincipals.isEmpty())
            return false;

        // Verificar si la identidad del peer y la autorizada son la misma
        return peerPrincipals.contains(this.authorizedIdentity);
    }

    public void initialize(PDPConfig config, String name, String id)
        throws InitializeException {

        // Lee la configuración del descriptor de despliegue del servicio
        this. authorizedIdentity =
            new GlobusPrincipal((String) config.getProperty(
                name, "authorizedIdentity"));
    }

    /* Handle de parade */
    public void close() throws CloseException {
        this. authorizedIdentity = null;
    }
```

```
}
```

Para poder usar el PDP codificado es necesario hacerlo explícito en el descriptor de seguridad del servicio:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <authz value="miAuthz:ejemplo.pdp.impl.MiPDP"/>
   ...
<securityConfig/>
```

Ahora es necesario configurar el PDP en el descriptor de despliegue del servicio e indicar la identidad válida, tal y como se ha codificado:

```
<service name="MiServicio"
            provider="Handler"
            style="document">
   ...
   <parameter name="securityDescriptor"
                     value="/...path-en-local-fs/-security-config.xml"/>
   <parameter name="miAuthz-authorizedIdentity"

value="/DC=fundacionparalaleyyelorden/DC=nait/OU=Driver/CN=Maiquel Nait"/>
   ...
</service>
```

El parámetro miAuthz-authorizedIdentity de la configuración se deriva colocando el prefijo usado para especificar el PDP en el descriptor de seguridad y la propiedad necesaria para que el PDP obtenga el parámetro de configuración.

5.3.3 Client-side

5.3.3.1 Configuración de las credenciales

Las credenciales se configuran igual que en el lado del servidor.

5.3.3.2 Configuración de los mecanismos de autorización

El elemento `<authz/>` es usado para determinar el mecanismo que se va a usar para contactar con el servidor.

Los valores que están soportados son:

- none. No se realiza ninguna autorización.
- self. El servidor tiene que estar ejecutándose con las mismas credenciales que el cliente.
- host. El servidor tiene que estar ejecutándose con las credenciales incluyendo el host en el que se está ejecutando.
- Otra. Se realizará la autorización según la identidad del cliente.

Por ejemplo:

```
<securityConfig xmlns="http://www.globus.org">
     ...
     <authz value="self"/>
     ...
```

```
  </securityConfig>
```

5.3.3.3 Configurando GSISecureConversation

El cliente también puede ser configurado para que utilice GSISecureConversation usando el elemento `<GSISecureConversation/>`. Los siguientes suplementos pueden añadirse como propiedades:

- `<integrity/>`. Fija la protección a nivel de firma.
- `<privacy/>`.Fija la protección a nivel del encriptación. También se realiza a nivel de firma.
- `<delegation value="tipo" />`. Determina el tipo de delegación que se debe realizar. El valor puede ser full o limited. Si este elemento no se usa no se realizará ninguna delegación.

El siguiente ejemplo ilustra la forma en que se podría configurar el descriptor de seguridad del cliente:

```
<securityConfig xmlns="http://www.globus.org">
     ...
     <GSISecureConversation>
         <privacy/>
         <delegation value="full"/>
     </GSISecureConversation>
     ...
  </securityConfig>
```

5.3.3.4 Configurando GSISecureMessage

Utilizando el parámetro GSISecureMessage con los siguientes elementos, el cliente utilizará dicha opción del sistema de seguridad de globus:

- `<integrity/>`. Fija la protección a nivel de firma.
- `<privacy/>`.Fija la protección a nivel del encriptación. También se realiza a nivel de firma.
- `<peer-credential value="path-fichero" />`. Determina el path al fichero con las credenciales a usar si la protección de privacidad es seleccionada.

El siguiente ejemplo muestra el uso de esta opción:

```
<securityConfig xmlns="http://www.globus.org">
     ...
     <GSISecureMessage>
         <integrity/>
     </GSISecureMessage>
     ...
  </securityConfig>
```

5.3.4 Descriptores de seguridad para recursos

La seguridad a nivel de recurso puede ser configurada usando un descriptor de seguridad que sobrescriba la configuración de seguridad del servicio y del contenedor. Para hacer un recurso 'seguro', es necesario implementar el interfaz `org.globus.wsrf.impl.security.SecureResource`. Este interfaz tiene un método que

devuelve una instancia de `org.globus.wsrf.impl.security.descriptor.ResourceSecurityDescriptor`. Si devolviese `null`, se asume que no existe descriptor para ese recurso.

El descriptor de seguridad del recurso es idéntico al de un servicio y expone la API con métodos set y get para todas las propiedades descritas en el apartado del descriptor de despliegue del servidor.

El siguiente fragmento de código crea un descriptor de seguridad de manera directa:

```
ResourceSecurityDescriptor desc = new ResourceSecurityDescriptor();
desc.setRejectLimitedProxy("true");
```

El siguiente fragmento de código crea un descriptor de seguridad desde un fichero:

```
ResourceSecurityConfig config = new
ResourceSecurityConfig("fileName");
config.init();
ResourceSecurityDescriptor desc = config.getSecurityDescriptor();
```

Existen dos atributos del descriptor de seguridad, credentials y gridmap, que son fácilmente especificables como objetos o como rutas a los ficheros de credenciales y grid-mapfile.

La caedna de autorización del servicio y los PDPs pueden ser especificados usando una lista separa por comas. En cada caso,si las propiedades están configuradas como ficheros o nombres de PDP será necesario usar la API con el helper `org.globus.wsrf.impl.security.descriptor.ResourceSecurityConfig` para cargar las clases. Las credenciales, grid-mapfile y PDPs especificadas en la cadena de autorización son cargados si la propiedad `initialized` está a `false` en el descriptor.

Por ejemplo, el siguiente fragmento de código cera un descriptor con el grid-mapfile y una cadena de autorización. Cuando config.init() es llamado, el grid-mapfile es cargado y una instancia de la cadena de autorización es creada. La información de la configuración se obtiene del descriptor de despliegue global. Para fijar la configuración de otro PDP es necesario hacer vía código:

```
ResourceSecurityDescriptor desc = new ResourceSecurityDescriptor();

desc.setGridMapFile("foo/bar/gridmap");
desc.setAuthz(
    "customAuthz:org.globus.some.customAuthz foo1:org.foo.barAuthz");

ResourceSecurityConfig config = new ResourceSecurityConfig(desc);

config.init();
```

Si el descriptor cambia, se puede forzar para que sea recargado fijando la propiedad `initilized` a `false`:

```
desc.setInitialized(false);
desc.setGridMapFile("foo/bar/newGridMap");
config.init();
```

También es posible asignar la identidad y el gridmap sin depender de ficheros:

```
desc.setInitialized(false);
GridMap map = new GridMap();
```

```
map.map("Some user DN", "userid");
desc.setGridMap(map);
```

La autorización también se puede realizar creando el grafo completo de objetos a partir de la API org.globus.wsrf.security de diversas formas, aunque si bien es cierto, no existe ninguna recomendación que se ciña al uso de ficheros y sí al de clases e interfaces suministrados.

5.3.5 Configuración de seguridad del contenedor

Además de las propiedades que han sido descriptas para los descriptores de seguridad, existen dos propiedades más que son exclusivas del contenedor.

- Cuando se usa GSISecureconversation se establece un contexto de seguridad. Una tarea se ejecuta cada diez minutos para borrar todos los contextos expirados de anteriores sesiones. Este intervalo puede ser configurado en el descriptor de seguridad del contenedor de la siguiente manera:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <context-lifetime value="666"/> <!—- milisegundos -->
   ...
</securityConfig>
```

- Para la seguridad a nivel de mensaje existe también un intervalo para el seguimiento de los mensajes recibidos con el propósito de establecer una protección contra la repetición de mensajes. Mensajes fuera de esa ventana serán rechazados unilateralmente y los mensajes que sí se encuentren dentro de ese intervalo serán comprobados entre sí a tra´ves del uso de identificadores únicos UUID. La configuración se realiza como sigue:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <replay-attack-interval-lifetime value="666"/>
   <!—- milisegundos -->
   ...
</securityConfig>
```

6. Caso de estudio

6.1 Objetivos

Como objetivos del caso de estudio se fijaron la implementación de un servicio web desplegable en el contenedor GT4 y configurar e investigar el entorno de seguridad, su versatilidad y posibilidades de configuración, así como evaluar el desarrollo de servicios e interfaces para implementaciones alternativas. Además, se ha contemplado una propuesta de mejora del producto basándose en estándares y revisiones de otras tecnologías que actualmente conforman la punta de lanza de la escena del desarrollo de software mundial.

6.2 Instalación y herramientas

6.2.1 GT4

Para el desarrollo del caso de estudio se procedió a la instalación del contenedor como copia local al usuario del proyecto en el entorno del laboratorio sobre el que se realizaron todos los ensayos y pruebas.

6.2.2 Entorno Integrado de Desarrollo

La codificación, empaquetamiento y despliegue se realizó con el soporte del IDE Eclipse SDK 3 y

el plugin aún en fase alfa para GT4 que forma parte de las herramientas Globus Build Tools (http://gsbt.sourceforge.net/) GT4IDE.

Eclipse SDK 3 ha permitido la codificación y edición de clases y ficheros de configuración así como la ejecución y depuración del código cliente realizado para el caso de estudio.

Eclipse SDK 3 es un entorno integrado de desarrollo, IDE, muy popular y libre que se haya en su mejor momento gracias, entre otras cosas, a su integración con plugins de terceros, arrebatando el mercado a las herramientas propietarias de fabricantes comerciales gracias al apoyo de la comunidad de desarrolladores y de grandes multinacionales que apuestan por este modelo como IBM, BEA, Nokia, Borland, Racional, Together, RedHat, SuSE y otros. El proyecto Eclipse lo conforman otras herramientas que se basan en un entorno integrado para construir aplicaciones avanzadas de escritorio así como la orientación a ingeniería del software y a nuevas tecnologías dentro del software.

6.2.3 Sistema

Es importante que el usuario de desarrollo esté dado de alta en el fichero grid-mapfile para poder realizar validaciones utilizando ese mismo fichero y comprobar que todo va bien.

6.3 Primeros pasos

Después de la configuración de todo el entorno y la búsqueda de la idoneidad en la configuración de las herramientas, el contenedor y el sistema se realizó una labor de aprendizaje en la que se centraron los esfuerzos en la asimilación de conceptos clave, modelo de componentes y funcionamiento del contenedor GT4.

Para ello se utilizó la documentación oficial de globus.org y el tutorial-guía del programador disponible en el mismo website.

En este punto destaca la falta de documentación concisa y clara que sirva de ayuda frente a nuevos usuarios de la tecnología así como la cantidad de imprecisiones debido al estado de desarrollo continuo del producto.

A pesar de los devenires de la situación, el primer servicio web fue desplegado con éxito rápidamente.

El código del servicio se muestra a continuación:

```
package es.ucm.dacya.gridway.wsrf.impl;

import java.rmi.RemoteException;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;

import es.ucm.dacya.gridway.wsrf.stubs.AddResponse;
import es.ucm.dacya.gridway.wsrf.stubs.GetValueRemote;

/**
  * Implementación de un servicio web para GT4
  */
public class GridWayService implements Resource, ResourceProperties {

      /* Propiedades del recurso */
      private ResourcePropertySet propSet;
```

```
	/* En el ejemplo, el recurso es un número entero */
	private int value;

	/* Constructor. Initializes RPs */
	public GridWayService()throws RemoteException {
		this.propSet = new
SimpleResourcePropertySet(GridWayQNames.RESOURCE_PROPERTIES);

		try {
			/* Inicialización de las propiedades */
			ResourceProperty valueRP = new
ReflectionResourceProperty(GridWayQNames.RP_VALUE, "Value", this);
			this.propSet.add(valueRP);

			setValue(0);
		} catch (Exception e) {
			throw new RuntimeException(e.getMessage());
		}
	}

	public int getValue() {
		return value;
	}

	public void setValue(int value) {
		this.value = value;
	}


	/* Requerido por el interface interface ResourceProperties */
	public ResourcePropertySet getResourcePropertySet() {
		return this.propSet;
	}

	// Método de prueba del servicio que realiza una suma sonre el recurso
	public AddResponse add(int cantidad) throws RemoteException {
		setValue(getValue() + cantidad);

		return new AddResponse();
	}
	// Método de consulta del valor del recurso
	public int getValueRemote(GetValueRemote params) throws RemoteException {
		return value;
	}
}
```

A continuación, según la recomendación del tutorial, se codificó una clase con las constantes y lo NS del servicio web:

```
package es.ucm.dacya.gridway.wsrf.impl;

import javax.xml.namespace.QName;

public class GridWayQNames {
	public static final String NS =
"http://www.dacya.ucm.es/wsrf/services/GridWayService";

	public static final QName RESOURCE_PROPERTIES = new QName(NS,
```

```
                    "GridWayResourceProperties");

      public static final QName RESOURCE_REFERENCE = new QName(NS,
                    "GridWayResourceReference");

      public static final QName RP_VALUE = new QName(NS, "Value");
}
```

El descriptor de despliegue quedó como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>

<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <service name="wsrf/services/GridWayService" provider="Handler"
use="literal" style="document">

            <!—- Clase del servicio -->
            <parameter name="className"
value="es.ucm.dacya.gridway.wsrf.impl.GridWayService"/>

            <!—- publicar y permitir acceso a todos los métodos especificados
-->
            <parameter name="allowedMethods" value="*"/>

            <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
            <parameter name="scope" value="Application"/>
            <parameter name="loadOnStartup" value="true"/>

      <!-- Ruta al descriptor del servicio web -->
      <wsdlFile>share/schema/gt4ide/GridWayService/GridWay_service.wsdl</wsdlFi
le>

      </service>
</deployment>
```

Como se puede observar, este descriptor no incluye seguridad y publica todos los métodos del servicio web desplegado.

El descriptor del servicio web es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="GridWayService"
    targetNamespace="http://www.dacya.ucm.es/wsrf/services/GridWayService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.dacya.ucm.es/wsrf/services/GridWayService"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.wsdl"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
```

```
ResourceProperties-1.2-draft-01.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-
draft-01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-
01.wsdl"
    location="../../wsrf/lifetime/WS-ResourceLifetime.wsdl" />


<types>
<xsd:schema
targetNamespace="http://www.dacya.ucm.es/wsrf/services/GridWayService"
    xmlns:tns="http://www.dacya.ucm.es/wsrf/services/GridWayService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

       <!—- Definición de los tipos de peticiones y respuestas -->

       <xsd:element name="add" type="xsd:int"/>
       <xsd:element name="addResponse">
             <xsd:complexType/>
       </xsd:element>

       <xsd:element name="getValueRemote">
             <xsd:complexType/>
       </xsd:element>
       <xsd:element name="getValueRemoteResponse" type="xsd:int"/>

       <!—- Definición de los tipos -->

       <xsd:element name="Value" type="xsd:int"/>

       <xsd:element name="GridWayResourceProperties">
       <xsd:complexType>
             <xsd:sequence>
                   <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
             </xsd:sequence>
       </xsd:complexType>
       </xsd:element>
</xsd:schema>
</types>

<!—- Definición de los mensajes -->

<message name="AddInputMessage">
       <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
       <part name="parameters" element="tns:addResponse"/>
</message>

<message name="GetValueRemoteInputMessage">
       <part name="parameters" element="tns:getValueRemote"/>
</message>
<message name="GetValueRemoteOutputMessage">
```

```
      <part name="parameters" element="tns:getValueRemoteResponse"/>
</message>


<portType name="GridWayPortType"
    wsrp:ResourceProperties="tns:GridWayResourceProperties">


      <!—- Inicio de la declaración de operaciones -->
      <operation name="add">
            <input message="tns:AddInputMessage"/>
            <output message="tns:AddOutputMessage"/>
      </operation>

      <operation name="getValueRemote">
            <input message="tns:GetValueRemoteInputMessage"/>
            <output message="tns:GetValueRemoteOutputMessage"/>
      </operation>

</portType>

</definitions>
```

Para completar el servicio web es necesaria la generación de stubs, operación integrada en el plugin GT4IDE que facilita esa tarea automática.

Esos stubs también son usados por el programa cliente, cuyo código queda como sigue:

```
package es.ucm.dacya.gridway.ws.cliente;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import es.ucm.dacya.gridway.wsrf.stubs.GetValueRemote;
import es.ucm.dacya.gridway.wsrf.stubs.GridWayPortType;
import es.ucm.dacya.gridway.wsrf.stubs.service.GridWayServiceAddressingLocator;


/**
 * Clase cliente del servicio web para GT4
 */
public class Main {

      public static void main(String[] args) throws Exception {
            GridWayServiceAddressingLocator instanceLocator = new
GridWayServiceAddressingLocator();

            int valor = Integer.parseInt(args[1]);
            EndpointReferenceType instanceEPR;

            String serviceURI = args[0];

            // Crear referencia al endpoint del servicio
            instanceEPR = new EndpointReferenceType();
            instanceEPR.setAddress(new Address(serviceURI));

            // Obtener referencia al interface del servicio (Stub)
            GridWayPortType gridWay =
```

```
instanceLocator.getGridWayPortTypePort(instanceEPR);

            // ejecutar operacion remota
            gridWay.add(valor);

            // obtener valor de retorno (Recurso)
            int retorno = gridWay.getValueRemote(new GetValueRemote());

            // Imprimir valor actual
            System.out.println("Valor: " + retorno);
      }
}
```

6.4 Despliegue del servicio

A continuación se empaquetó el servicio web siguiendo la especificación de GT4 en un Globus Application Resource o GAR y se desplegó en el contenedor.

```
[jm05@hydrus ~]$ globus-deploy-gar GridWay.gar
Deploying gar file...

  Deploying gar with profile: <default>
  Created dir: /home/jm05/globus/share/globus_wsrf_common/tmp/gar
  Skipping fileset for directory /home/jm05/globus/share/globus_wsrf_common/tmp.
It is empty.
  Created dir: /home/jm05/globus/share/globus_wsrf_common/tmp/gar/schema
  Created dir: /home/jm05/globus/share/globus_wsrf_common/tmp/gar/etc
  Created dir: /home/jm05/globus/share/globus_wsrf_common/tmp/gar/bin
  Created dir: /home/jm05/globus/share/globus_wsrf_common/tmp/gar/docs
  Created dir: /home/jm05/globus/share/globus_wsrf_common/tmp/gar/share
  Created dir: /home/jm05/globus/share/globus_wsrf_common/tmp/gar/lib
  Expanding: /home/jm05/eclipse/workspace/GridWay/GridWay.gar into
/home/jm05/globus/share/globus_wsrf_common/tmp/gar
  Skipping fileset for directory /home/jm05/globus/etc/GridWay. It is empty.
  Skipping fileset for directory /home/jm05/globus. It is empty.
  Skipping fileset for directory /home/jm05/globus/bin. It is empty.
  Skipping fileset for directory /home/jm05/globus/share/licenses. It is empty.
  deploying server config...
  deploying JNDI config...
  Deleting directory /home/jm05/globus/share/globus_wsrf_common/tmp/gar

Deploy successful
```

En contraste con otras tecnologías que definen paquetes similares a los GAR y que consisten en un conjunto de descriptores, clases y archivos (por ejemplo el estándar J2EE, ahora Java EE, define varios tipos de distribuciones: EJB, WAR, EAR), y que se distribuyen en un directorio local, sin interferir en el contenedor, un GAR se despliega copiando varios ficheros en la instalación del contenedor, incorporando JARs a las librerías y copiando sólo los descriptores de despliegue y de servicio web en un directorio específicamente aislado a modo de registro.

6.5 Seguridad en el caso de estudio

Para cubrir el objetivo primordial del caso de estudio se procedió a investigar la incorporación de mecanismos de seguridad a un servicio web básico.

Dado que se trataba de un contenedor de pruebas, el estudio se ha centrado en la seguridad referida al servicio y al recurso, dejando la seguridad del contenedor como un paso de madurez

del sistema correspondiente al paso a producción de la instalación de GT4 en un entorno real.

6.5.1 Estableciendo una comunicación segura usando SSL

En el primer *stage* del servicio no existía ninguna garantía de seguridad en la ejecución del servicio ni en el paso de mensajes entre el cliente y el contenedor en la secuencia de petición y respuesta que se realiza al invocar a un servicio web.

Para que el contenedor pueda aceptar peticiones usando el protocolo HTTPS es necesario iniciarlo sin la opción –nosec. De esta manera, las direcciones de los servicios se transforman para comenzar con https.

El servicio desplegado no necesita incorporar ninguna información extra ni código suplementario para soportar comunicaciones HTTPS, pero el código del cliente deja de funcionar mostrando mensajes aparentemente sin sentido.

Se trata de la librería para servicios web axis de apache, que necesita algún tipo de inicialización no documentada en su integración con GT4 y que registra las factorías necesarias para el protocolo HTTPS. El código del cliente queda como sigue:

```
package es.ucm.dacya.gridway.ws.cliente;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import es.ucm.dacya.gridway.wsrf.stubs.GetValueRemote;
import es.ucm.dacya.gridway.wsrf.stubs.GridWayPortType;
import es.ucm.dacya.gridway.wsrf.stubs.service.GridWayServiceAddressingLocator;
import org.globus.axis.util.Util;

/**
 * Clase cliente del servicio web para GT4
 */
public class Main {

      public static void main(String[] args) throws Exception {
            //init https factory
            Util.registerTransport();

            GridWayServiceAddressingLocator instanceLocator = new
GridWayServiceAddressingLocator();

            int valor = Integer.parseInt(args[1]);
            EndpointReferenceType instanceEPR;

            String serviceURI = args[0];

            // Crear referencia al endpoint del servicio
            instanceEPR = new EndpointReferenceType();
            instanceEPR.setAddress(new Address(serviceURI));

            // Obtener referencia al interface del servicio (Stub)
            GridWayPortType gridWay =
instanceLocator.getGridWayPortTypePort(instanceEPR);

            // ejecutar operacion remota
            gridWay.add(valor);

            // obtener valor de retorno (Recurso)
            int retorno = gridWay.getValueRemote(new GetValueRemote());
```

```
            // Imprimir valor actual
            System.out.println("Valor: " + retorno);
      }
}
```

6.5.2 *Descriptor de seguridad en el servicio*

Una vez que se alcanzaron los objetivos mínimos de seguridad entre dos puntos que establecen una comunicación en una red, como es hasta la fecha la inclusión de SSL en el protocolo, esta vez HTTPS, se procedió a investigar qué tipos de seguridad plantea GT4 y cómo se utilizarían en un servicio web básico.

La primera sorpresa fue que este descriptor no tiene por qué distribuirse con el servicio, sino que es referenciado por el descriptor de despliegue como fichero en el sistema de archivos local (como aclaración hay que destacar que el fichero no tiene por qué ser estrictamente local, sino que debe ser accesible desde el sistema de archivos, pudiendo encontrarse en una unidad de red compartida, extraíble o cualquiera que sea el mecanismo de almacenamiento).

Una de las opciones es incluir una ruta relativa al descriptor y distribuirlo en el propio GAR. Esto, que parecería lo lógico, no es posible por varias razones:

- Habría que conocer si el contenedor interpreta el directorio . como local al servicio y no al contenedor.
- La operación de despliegue no garantiza que el descriptor de seguridad sea copiado a un directorio determinado.
- Es imposible saber de antemano cuál será el directorio, dentro de la instalación de globus, sobre el que se desplegará el servicio web.

La primera prueba consiste en la verificación de que el despliegue de un servicio con parámetros de seguridad es desplegado correctamente.

El descriptor de despliegue queda como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>

<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <service name="wsrf/services/GridWayService" provider="Handler"
use="literal" style="document">

            <parameter name="className"
value="es.ucm.dacya.gridway.wsrf.impl.GridWayService"/>
            <parameter name="allowedMethods" value="*"/>
            <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
            <parameter name="scope" value="Application"/>
            <parameter name="loadOnStartup" value="true"/>
            <!-- Ruta del descriptor de seguridad -->
            <parameter name="securityDescriptor"
value="/home/jm05/descriptorSeguridad.xml"/>

      <wsdlFile>share/schema/gt4ide/GridWayService/GridWay_service.wsdl</wsdlFi
le>

      </service>
</deployment>
```

Y el descriptor de seguridad de ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>

<securityConfig xmlns="http://www.globus.org">
  <auth-method>
      <none/>
   </auth-method>
</securityConfig>
```

6.5.3 Seguridad en el cliente

Para reforzar la seguridad en el cliente es necesario introducir código en la clase propuesta para que sean verificadas las condiciones de autenticación y autorización. A continuación se detalla el código comentando las modificaciones:

```
package org.globus.clients.HelloWorldService;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import es.ucm.dacya.gridway.wsrf.stubs.GetValueRemote;
import es.ucm.dacya.gridway.wsrf.stubs.GridWayPortType;
import es.ucm.dacya.gridway.wsrf.stubs.service.GridWayServiceAddressingLocator;
import org.globus.axis.util.Util;

// imports para incorporar seguridad
import javax.xml.rpc.Stub;
import org.globus.wsrf.impl.security.authentication.Constants;
import org.globus.gsi.GSIConstants;
import org.globus.wsrf.impl.security.authorization.SelfAuthorization;
import org.globus.wsrf.impl.security.authorization.HostAuthorization;
import org.globus.wsrf.impl.security.authorization.NoAuthorization;


/**
 * Clase cliente del servicio web para GT4
 */
public class Main {

    public static void main(String[] args) {
       //init https factory
       Util.registerTransport();

         try {
              GridWayServiceAddressingLocator instanceLocator = new
GridWayServiceAddressingLocator();

              int valor = Integer.parseInt(args[1]);
              EndpointReferenceType instanceEPR;

              String serviceURI = args[0];

              // Crear referencia al endpoint del servicio
              instanceEPR = new EndpointReferenceType();
```

```
            instanceEPR.setAddress(new Address(serviceURI));

            // Obtener referencia al interface del servicio (Stub)
            GridWayPortType gridWay =
instanceLocator.getGridWayPortTypePort(instanceEPR);


            // Fija los parámetros de seguridad
            ((Stub) gridWay)._setProperty(Constants.GSI_TRANSPORT,
Constants.SIGNATURE);
            ((Stub) gridWay)._setProperty(Constants.GSI_TRANSPORT,
Constants.ENCRYPTION);
            ((Stub) gridWay)._setProperty(Constants.AUTHORIZATION,
HostAuthorization.getInstance());

            // ejecutar operacion remota
            gridWay.add(valor);

            // obtener valor de retorno (Recurso)
            int retorno = gridWay.getValueRemote(new GetValueRemote());

            // Imprimir valor actual
            System.out.println("Valor: " + retorno);
      }
}
```

También es posible indicar cómo se va a proceder a nivel de seguridad en un descriptor del cliente en lugar de tenerlo que codificar:

```
package org.globus.clients.HelloWorldService;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import es.ucm.dacya.gridway.wsrf.stubs.GetValueRemote;
import es.ucm.dacya.gridway.wsrf.stubs.GridWayPortType;
import es.ucm.dacya.gridway.wsrf.stubs.service.GridWayServiceAddressingLocator;
import org.globus.axis.util.Util;

// imports para incorporar seguridad
import javax.xml.rpc.Stub;
import org.globus.wsrf.impl.security.authentication.Constants;
import org.globus.gsi.GSIConstants;
import org.globus.wsrf.impl.security.authorization.SelfAuthorization;
import org.globus.wsrf.impl.security.authorization.HostAuthorization;
import org.globus.wsrf.impl.security.authorization.NoAuthorization;


/**
 * Clase cliente del servicio web para GT4
 */
public class Main {

    public static void main(String[] args) {
       //init https factory
       Util.registerTransport();

        try {
```

```
            GridWayServiceAddressingLocator instanceLocator = new
GridWayServiceAddressingLocator();

            int valor = Integer.parseInt(args[1]);
            EndpointReferenceType instanceEPR;

            String serviceURI = args[0];

            // Crear referencia al endpoint del servicio
            instanceEPR = new EndpointReferenceType();
            instanceEPR.setAddress(new Address(serviceURI));

            // Obtener referencia al interface del servicio (Stub)
            GridWayPortType gridWay =
instanceLocator.getGridWayPortTypePort(instanceEPR);


            // Inicializar según el descriptor de seguridad
            ((Stub) gridWay)._setProperty(Constants.CLIENT_DESCRIPTOR_FILE,
"descriptorSeguridadCliente.xml");

            // ejecutar operacion remota
            gridWay.add(valor);

            // obtener valor de retorno (Recurso)
            int retorno = gridWay.getValueRemote(new GetValueRemote());

            // Imprimir valor actual
            System.out.println("Valor: " + retorno);
      }
}
```

Siendo el descriptor de seguridad del cliente como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>

<securityConfig xmlns="http://www.globus.org">

    <authz value="host"/>

    <GSITransport>
        <integrity/>
        <privacy/>
    </GSITransport>
</securityConfig>
```

Los valores del método de autorización pueden ser cambiados fácilmente a `self` o `identity`, sin duda los más interesantes de los propuestos.

6.5.4 Seguridad en el servidor

Como ya se vio anteriormente, la seguridad en el lado del servidor se realiza especificando un descriptor de seguridad referenciado en el descriptor de despliegue del servicio web. La línea de configuración en el dicho descriptor de despliegue del servicio web es:

```
<parameter name="securityDescriptor" value="ruta al descriptor">
```

Si es necesario realizar la autorización de la forma `host`, también es importante que esta línea esté en el descriptor de despliegue como configuración:

```
<parameter name="hostAuthz-url" value="nombre del host autorizado">
```

En el caso del ejemplo, el descriptor de seguridad queda como sigue:

```
<securityConfig xmlns="http://www.globus.org"
xmlns:hello="http://www.globus.org/namespaces/wsrf/HelloWorldService">

<authz value="none"/>

<!—- Autorización para el método add. Con encriptación -->
<method name="gridWay:add">
  <auth-method>
    <GSITransport>
      <protection-level>
        <integrity/>
        <privacy/>
      </protection-level>
    </GSITransport>
  </auth-method>
</method>

<!—- Autorización para los demás métodos -->
<auth-method>
  <GSITransport>
    <protection-level>
      <integrity/>
      <privacy/>
    </protection-level>
 </GSITransport>
</auth-method>

</securityConfig>
```

En el caso de que deseemos implementer algún otro mecanismo de autenticación, deberá ser indicado en el descriptor. A continuación se presentan algunos ejemplos:

o Self.

```
<authz
value="selfAuthz:org.globus.wsrf.impl.security.authorization.SelfAuthorization"/
>
```

o Gridmap.

```
<authz
value="gridmap:org.globus.wsrf.impl.security.authorization.GridMapAuthorization"
/>
<gridmap value="/ruta-local/grid-mapfile"/>
```

o SAML

```
<authz
value="samlAuthz:org.globus.wsrf.impl.security.authorization.SAMLAuthorizationCa
llout"/>
```

Para implementar autenticación basada en grid-mapfile es importante que el fichero indicado en la configuración de seguridad también esté en el sistema de ficheros o accesible por el contenedor siguiendo la ruta especificada. El fichero, evidentemente, debe contener una lista de DNs que se correspondan con los usuarios del sistema. Esto hará que el DN del cliente que realiza la petición, tomado del certificado proxy, sea resuelto a través de los usuarios del sistema en la propia llamada. Además, sirve de lista de control de acceso (Access Control List, ACL) de baja resolución.

6.5.5 Ejemplo complejo

Finalmente, fue desarrollado un ejemplo más complejo en el que se introdujo seguridad en la autorización GSISecureConversation y autenticación gridmap.

El descriptor de seguridad del servicio quedaría:

```
<securityConfig xmlns="http://www.globus.org">

  <auth-method>
    <GSISecureConversation>
      <protection-level>
        <integrity/>
        <privacy/>
      </protection-level>
    </GSISecureConversation>
  </auth-method>

  <authz value="gridmap"/>
  <gridmap value="/etc/grid-security/grid-mapfile"/>

</securityConfig>
```

El cliente necesita ajustar los parámetros de seguridad en el descriptor:

```
<securityConfig xmlns="http://www.globus.org">

  <authz value=”host”>

  <GSISecureConversation>
     <privacy/>
     <delegation value=”full”/>
  </GSISecureConversation>

</securityConfig>
```

7. Conclusiones

Después del proceso de investigación, del caso de estudio que ha incluido el desarrollo y el buceo en conceptos novedosos, tanto a nivel de tecnología como de producto (es importante recordar que la primera versión estable de GT4 es del año actual), se presentan las conclusiones en las que se exponen también sugerencias a cerca del rumbo que, según los autores, podría tomar el contenedor de servicios web Globus Toolkit.

7.1 Modelo de componentes

El modelo de componentes ha sido diseñado a partir de ciertas premisas que tienen sentido en el contexto de la tecnología grid. Es cierto que el planteamiento con respecto al ciclo de vida sí parece el adecuado, pero su implementación es demasiado similar a otros modelos de componentes de tecnologías ya clásicas como EJB dentro del framework J2EE (Ahora Java EE) y que se han mostrado ciertamente ineficientes debido a su complejidad, tanto en su desarrollo,

como en el despliegue y la ejecución, derivando en problemas de escalabilidad y mantenibilidad.

De hecho, de los cuatro tipos de EJBs de la especificación, se reproducen dos con patrones similares: Stateless Session Beans y Stateful Session Beans.

Si bien es cierto que uno de los requisitos más relevantes es la necesidad de desplegar los servicios de manera que se puedan distribuir, la experiencia de otros estándares cuya comunidad es de un tamaño enorme ha demostrado que lo que en teoría es una implementación que alcanza los objetivos propuestos, no tiene gran recorrido a medio plazo y es necesario cambiar (Actualmente el estándar de EJB está siendo redefinido en lo que será la versión 3).

7.1.1 El coste de la complejidad

La documentación del producto, la disponible de otras fuentes, tutoriales y demás recursos muestran ejemplos de servicios web sobre globus que son difíciles de entender. Esto es, los ejemplos apenas muestran funcionalidades de computación y están plagados de código y llamadas a las APIs de Globus, demostrando la complejidad de desplegar cualquier distribución en el contenedor.

En general, una arquitectura compleja es siempre una mala idea. La complejidad implica que es necesario emplear tiempo en aprender, cuesta más desarrollar y mantener la aplicación que en una arquitectura sencilla. Una lista de lo que nos encontramos en GT4 es:

- Cada línea de código que es escrita innecesariamente debe ser probada y mantenida a lo largo de la vida de la aplicación. Este esfuerzo podría emplearse mejor en la codificación de otros servicios o en mejorar los existentes.
- Las arquitecturas complejas se han probado ciertamente ineficientes. Demasiadas capas, demasiada abstracción, demasiadas llamadas remotas: todo contribuye a un rendimiento pobre.
- Un código complicado es difícil de entender, haciendo aún más difícil mantenerlo.
- Un código complicado implica incrementar la propensión a introducir errores, consumiendo recursos innecesariamente.

Lo que de verdad muestran los ejemplos es que un servicio web en GT4 necesita demasiado código ajeno al cómputo, al objetivo real del servicio.

7.1.2 Principales carencias en el diseño del modelo de componentes

Algo que es innegable es que las aplicaciones necesitan un backbone de infraestructura sobre el que apoyarse. En este caso es necesario disponer de ciertos servicios que el contenedor presta a las aplicaciones desplegadas y que gestiona el ciclo de vida de los recursos y servicios.

La gestión de objetos y servicios, de hecho, tienen, prácticamente implícito, la existencia de un contenedor que los gestiona. Es posible hacer el mismo trabajo son un contenedor, pero en ese caso sí que estaremos cayendo en una falacia sin sentido: nos llevará a construir Singletons y factorías a medida así como a gestionar los parámetros de configuración de manera ineficiente e inconsistente. Unos preferirán la base de datos, otros ficheros XML, etc…

En este caso, GT4 añade cierto orden a este potencial caos ya que proporciona una capa de servicios bien definida, una solución para la gestión del ciclo de vida parcial… Es parcial puesto que es el cliente el que interacciona con los servicios para crear recursos con estado, añadiendo complicaciones al código cliente.

En otros términos, debido a las dependencias del código del servicio, este es muy difícil de probar. Cuanto más complejo sea, más tiempo se requerirá para conocer si existen errores o no puesto que será necesario desplegar el servicio y comprobar una por una las funcionalidades

con sus opciones, argumentos y excepciones para cerciorarnos que estamos desplegando una aplicación de calidad.

En resumen, el servicio debe ser diseñado y desarrollado para ser utilizado en GT4, incrementando en muchos enteros el esfuerzo necesario para realizar la misma tarea en una arquitectura más simple. La situación ideal no debería necesitar ni una sola línea de código que dependa en absoluto de GT4, siendo los descriptores bien diseñados para hacerlo de forma declarativa y servicio grid.

7.1.3 Inversion of Control, IoC

El deseo de poder evitar el código que dependa del contenedor no es algo irreal, aunque muchas de nuestras clases sigan teniendo dependencias y colaboradores. De hecho, lo normal es que dichos objetos realicen una operación de *lookup* para gestionar sus recursos y dependencias, por ejemplo para tener una referencia a otro servicio y satisfacer, en cierto modo, dichas dependencias.

Llegar a satisfacer las dependencias de un objeto son tener que realizar una operación de *lookup* sólo es posible, hasta la fecha, gracias a la magia del patrón Inversion of Control y de su particularización: Dependency Injection.

Se trata de dos conceptos clave que se basan en el principio de Hollywood: “No nos llames, ya te llamamos nosotros”, y que transfiere la responsabilidad de enlazar los objetos y sus colaboradores eliminando las operaciones de *lookup* en el código.

Las dos estrategias de IoC son:

- Dependency Lookup: el contenedor provee de llamadas a los componentes para que estos obtengan un contexto y puedan tomar la configuración, así como referencias a otros objetos, servicios y recursos. Es el caso de GT4.
- Dependency Injection: los componentes muestran un interfaz clásico sin dependencias en el contenedor. Es el contenedor el encargado de crear los objetos y enlazarlos a través de métodos set y constructores. Es necesario suministrarle al contenedor la configuración de las instancias de las clases necesarias para contextualizar la aplicación.

Un ejemplo ficticio de lo que sería un servicio web sencillo en GT4 si soportase Dependency Injection:

```
package es.ucm.dacya.gridway.wsrf.impl;

/**
  * Implementación de un servicio web para GT4
  */
public class GridWayImpl implements GridWay {

	/* En el ejemplo, el recurso es un número entero */
	private int value;

	public int getValue() {
		return value;
	}

	public void setValue(int value) {
		this.value = value;
	}

	// Método de prueba del servicio que realiza una suma sobre el recurso
	public void add(int cantidad) {
		value += cantidad;
```

```
        }
}
```

El servicio no contiene código del contenedor. Se supone que los métodos implementados del interfaz son todos los de la clase, el código del interfaz se omite por irrelevante para el ejemplo.

Una configuración podría ser:

```
<bean id="gridway" class="org.globus.wsrf.ProxyFactoryBean">
       <property name="serviceInterface">
              <value>es.ucm.dacya.GridWay </value>
       </property>
       <property name="serviceUrl">
              <value>http://www.dacya.ucm.es/myservice</value>
       </property>
       <property name="security">
              <value>…</value>
       </property>
       <property name="impl" ref="gridwayImpl"/>
</bean>

<bean id="gridwayImpl" class="es.ucm.dacya.GridWayImpl">
       <property name="value"><value>0</value></property>
</bean>
```

De esta forma, nos hemos inventado un wrapper que nos ayuda a configurar el servicio en GT4 sin que el código ni el diseño del servicio se vea afectado siendo:

- Más fácil de diseñar, desarrollar y probar, luego mucho más barato en recursos.
- Totalmente independiente y desacoplado del contenedor. Es la configuración de la aplicación la que contextualiza su despliegue en GT4, no el código del propio servicio.
- Es el contenedor el que trabaja para la aplicación, optimizando recursos y evitando complicar la arquitectura innecesariamente.
- Se promueve el diseño con interfaces, una buena práctica que parece imposible que no sea así en el GT4 actual.
- En caso de cambio de la arquitectura, el modo de configuración, etcétera, el servicio no debe ser recodificado, probado ni adaptado, tan solo reconfigurado ya que no existen dependencias con respecto a las librerías originales del producto.

Evidentemente, el ejemplo es bastante incompleto en cuanto a descripción de las operaciones, que ahora se realiza mediante descriptores pero que se podría hacer mediante introspección de la clase en base al interfaz del servicio, la seguridad, que puede añadirse como parte de la configuración del servicio y completar la especificación propuesta, etcétera. Aún así, el ejemplo muestra la capacidad de simplificación de esta aproximación y su potencia, así como los beneficios tangibles e irrefutables que conlleva.

7.2 Configuración

Otro de los caballos de batalla de GT4 es la configuración.

Debido a la aglutinación de estándares, integración con APIs, la adopción de los servicios web, tecnología redefiniéndose prácticamente cada año, y lo que se intuye como sinergia dentro de la comunidad JAVA con respecto a ciertas formas de operación que se consuman en la composición de la distribución GAR, nos encontramos con un entorno de configuración

heterogéneo en el que aún faltan cosas por consolidar; algunas ciertamente muy relevantes.

7.2.1 Despliegue

Una de las novedades que el framework J2EE introdujo en su concepción fue la distribución de aplicaciones en un formato nuevo, WAR, que respondía a dos necesidades:

1. La capacidad de gestionar la configuración del software en las aplicaciones desplegadas.
2. La necesidad de aislar en un mismo entorno de ejecución dos aplicaciones independientes.

En el primer caso, el fichero incluyó las clases de la aplicación así como un descriptor. En el interior del fichero se podía observar la estructura de la aplicación, con lo que se podía saber qué había y qué no había en dicha versión.

En el segundo fue algo más complejo: se instauraron varios niveles de classloaders en los que el espacio de nombres era jerárquico, siendo el nivel hoja el de la propia aplicación. Así no habría contaminación entre aplicaciones independientes.

Observando el comportamiento de GT4, es complicado deducir que alguno de estos objetivos haya sido propuesto en algún momento.

En primer lugar el fichero GAR consta de un JAR con las clases y los descriptores. El JAR es colocado en un directorio con más JARs, luego no parece que pertenezca a un nivel independiente y los descriptores a un directorio privado. El contenedor carga todas las clases de los JARs instalados y luego configura los servicios según la batería de descriptores disponibles.

7.2.2 Seguridad

Sin duda, una de las estrellas de GT4.

A pesar del gran esfuerzo que se ha realizado para completar el abanico de posibilidades que ofrece el producto todavía hay algunos ‘peros’ y puntos mejorables.

En primer lugar, dada la complejidad del sistema de autenticación y autorización, parece que se han olvidado del viejo mecanismo de las listas de control de acceso (ACLs). Parece que es muy complicado configurar un servicio al que no puedan acceder las mismas personas con diferentes permisos. Es decir, unos pueden ejecutar un método y otros no. Para ello es necesario codificar nuestro propio sistema de seguridad o extender la granularidad de los grid-mapfiles casi para cada usuario.

Tampoco es posible el empleo de expresiones regulares en el descriptor, técnica muy útil en otras tecnologías.

Otro de los problemas es la referencia al descriptor de seguridad desde el descriptor de despliegue. No se entiende que se trate de un fichero separado de la distribución GAR y que deba estar ubicado en el sistema de archivos plana y llanamente. Resulta insuficiente e inconsistente, difícil de mantener, no acorde con el concepto de contenedor y acción de despliegue propiamente.

7.2.3 AOP

Desde hace muy poco tiempo está sonando este nuevo enfoque de funcionalidades que complementa la programación orientada a objetos y que dice llamar programación orientada a aspectos (Aspect-Oriented Programming).

Se basa enun principio muy loable que es la eliminación de la repetición de código de las clases, principalmente el de las funcionalidades ortogonales clásicas: seguridad, persistencia, logging, transacciones, etcétera.

El típico ejemplo es el de las trazas de un programa. Lo normal es que codifiquemos un sistema basado en un singleton que centralice el sistema de log y que sea referenciado por las demás clases cuando creamos conveniente escribir una línea en el fichero o sistema de log. Así, cada clase tendrá líneas refiriéndose al singleton. Esto implica que, si queremos reutilizar cualquiera de las clases no es posible sin llevar consigo el singleton de las trazas. Si deseamos eliminar las trazas debeomos modificar toda la base de clases. Si deseamos cambiar la llamada a la función que escribe en el log indicando, por ejemplo, la severidad del mensaje, debemos modificar todo el código…

AOP propone lo siguiente: un mecanismo de clases Proxy de intercepción que son configuradas basándonos en los principios de IoC y dependency injection tal que las clases no deben contener tal código.

Supongamos que tenemos la clase del ejemplo anterior y que deseamos aplicar diferentes políticas de seguridad a los métodos. Existiría un interceptor que sería invocado de manera transparente para el cliente y configurada en el contenedor que dispararía el mecanismo de seguridad oportuno de manera declarativa (en la configuración).

7.3 Visión

En general, GT4 y las tecnologías grid están en un periodo de incubación interesante pero que aún necesita incorporarse al tren de alta velocidad que se está convirtiendo la comunidad empresarial actual, con J2EE a la cabeza en el que los impulsos de la comunidad de desarrolladores se ha tornado un factor decisivo en contraposición a la competición de productos anterior. Este punto de inflexión que ha sido la segunda mitad del año 2004 y el primer trimestre del 2005 no parece haber tocado a globus ni a su grupo de desarrollo de software, denotando un avance lento que puede garantizar la permanencia en escena si se realiza el trabajo con calidad o el olvido si no se torna la mirada hacia el exterior, hacia los hermanos mayores.

8. Bibliografía y referencias

- Servicios Web: http://www.w3.org/2002/ws/
- www.globus.org
- www.ibmdeveloperworks.com
- www.springframework.org
- www.aopalliance.org
- java.sun.com
- The Globus Toolkit 4 Programmer's Tutorial por Borja Sotomayor. http://gdp.globus.org/gt4-tutorial/

9. Palabras clave

- Globus Toolkit 4
- Sistemas Informáticos
- Grid
- Java
- Servicios web
- Descriptor de seguridad
- Computación distribuida